Student : Esteban Ordenes

Post Graduate Program in Data Science and Business Analytics

PGP-DSBA-UTA-Dec20-A

Background & Context

AllLife Bank is a US bank that has a growing customer base. The majority of these customers are liability customers (depositors) with varying sizes of deposits. The number of customers who are also borrowers (asset customers) is quite small, and the bank is interested in expanding this base rapidly to bring in more loan business and in the process, earn more through the interest on loans. In particular, the management wants to explore ways of converting its liability customers to personal loan customers (while retaining them as depositors).

A campaign that the bank ran last year for liability customers showed a healthy conversion rate of over 9% success. This has encouraged the retail marketing department to devise campaigns with better target marketing to increase the success ratio.

You as a Data scientist at AllLife bank have to build a model that will help the marketing department to identify the potential customers who have a higher probability of purchasing the loan.

Objective

- To predict whether a liability customer will buy a personal loan or not.
- Which variables are most significant.
- Which segment of customers should be targeted more.

Data Dictionary

- ID: Customer ID
- Age: Customer's age in completed years
- Experience: #years of professional experience
- Income: Annual income of the customer (in thousand dollars)
- ZIP Code: Home Address ZIP code.
- Family: the Family size of the customer
- CCAvg: Average spending on credit cards per month (in thousand dollars)
- Education: Education Level. 1: Undergrad; 2: Graduate;3: Advanced/Professional
- Mortgage: Value of house mortgage if any. (in thousand dollars)
- Personal_Loan: Did this customer accept the personal loan offered in the last campaign?
- Securities_Account: Does the customer have securities account with the bank?
- CD_Account: Does the customer have a certificate of deposit (CD) account with the bank?
- Online: Do customers use internet banking facilities?

- CreditCard: Does the customer use a credit card issued by any other Bank (excluding All life Bank)?

Load Libraries

In [513…

```
import pandas as pd
import numpy as np
import seaborn as sns
import math
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

import warnings
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
from statsmodels.stats.outliers_influence import variance_inflation_factor

warnings.filterwarnings('ignore')

# Removes the limit from the number of displayed columns and rows.
# This is so I can see the entire dataframe when I  print it
pd.set_option("display.max_columns", None)
# pd.set_option('display.max_rows', None)
pd.set_option("display.max_rows", 200)

# To build linear model for statistical analysis and prediction
import statsmodels.stats.api as sms
```

Load the Dataset

In [514…

```
data = pd.read_csv("Loan_Modelling.csv")

# copying data to another varaible to avoid any changes to original data
df = data.copy()

df.head()
```

Out[514…

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_Loan	Securi
0	1	25	1	49	91107	4	1.60	1	0	0	
1	2	45	19	34	90089	3	1.50	1	0	0	
2	3	39	15	11	94720	1	1.00	1	0	0	
3	4	35	9	100	94112	1	2.70	2	0	0	
4	5	35	8	45	91330	4	1.00	2	0	0	

In [514…

```
df.tail()
```

Out[514…

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_Loan

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_Loan
4995	4996	29	3	40	92697	1	1.90	3	0	0
4996	4997	30	4	15	92037	4	0.40	1	85	0
4997	4998	63	39	24	93023	2	0.30	3	0	0
4998	4999	65	40	49	90034	3	0.50	2	0	0
4999	5000	28	4	83	92612	3	0.80	1	0	0

Check the shape of the dataset

In [514…
```
print(f"There are {df.shape[0]} rows and {df.shape[1]} columns.")
```

```
There are 5000 rows and 14 columns.
```

`ID` is just an index for the data entry. This column will not be a significant factor in determining customers who have a higher probability of purchasing the loan.

`ZIPCode` this could contain a lot of zip code information. We can check check how many individual zip codes there are. If they are too many, we can process this column to extract group information.

`CCAvg` , as defined in the data dictionary represent thousands of dollars. We may need to convert these values.

In [514…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   ID                  5000 non-null   int64  
 1   Age                 5000 non-null   int64  
 2   Experience          5000 non-null   int64  
 3   Income              5000 non-null   int64  
 4   ZIPCode             5000 non-null   int64  
 5   Family              5000 non-null   int64  
 6   CCAvg               5000 non-null   float64
 7   Education           5000 non-null   int64  
 8   Mortgage            5000 non-null   int64  
 9   Personal_Loan       5000 non-null   int64  
 10  Securities_Account  5000 non-null   int64  
 11  CD_Account          5000 non-null   int64  
 12  Online              5000 non-null   int64  
 13  CreditCard          5000 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 547.0 KB
```

- All feature are numeric(either int64 or float64), but some of these represent categorical values.
- `ZIPCode` is a numerical variable, but it should be treated as categorical.
- `Education` is a numeric ordinal variable and could be represented as categorical
- `Personal_Loan` , `Securities_Account` , `CD_Account` , `Online` , `CreditCard` are numeric, but represent boolean (true/false) values.

Fixing the data types

- Convert `ZIPCode` and `Education` to categorical variables.

In [514…
```
df["ZIPCode"] = df["ZIPCode"].astype("category")
df["Education"] = df["Education"].astype("category")
```

In [514…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype   
---  ------              --------------  -----   
 0   ID                  5000 non-null   int64   
 1   Age                 5000 non-null   int64   
 2   Experience          5000 non-null   int64   
 3   Income              5000 non-null   int64   
 4   ZIPCode             5000 non-null   category
 5   Family              5000 non-null   int64   
 6   CCAvg               5000 non-null   float64 
 7   Education           5000 non-null   category
 8   Mortgage            5000 non-null   int64   
 9   Personal_Loan       5000 non-null   int64   
 10  Securities_Account  5000 non-null   int64   
 11  CD_Account          5000 non-null   int64   
 12  Online              5000 non-null   int64   
 13  CreditCard          5000 non-null   int64   
dtypes: category(2), float64(1), int64(11)
memory usage: 507.3 KB
```

Check the duplicate data. And if any, we should remove it.

In [514…
```
df[df.duplicated()].count()
```

Out[514…
```
ID                    0
Age                   0
Experience            0
Income                0
ZIPCode               0
Family                0
CCAvg                 0
Education             0
Mortgage              0
Personal_Loan         0
Securities_Account    0
CD_Account            0
Online                0
CreditCard            0
dtype: int64
```

- There are no duplicate values in the dataset

Check for missing values

In [514…
```
data.isnull().sum()
```

Out[514…
```
ID                    0
Age                   0
Experience            0
Income                0
ZIPCode               0
Family                0
CCAvg                 0
```

```
Education             0
Mortgage              0
Personal_Loan         0
Securities_Account    0
CD_Account            0
Online                0
CreditCard            0
dtype: int64
```

- There are no missing values in the dataset

Statistical summary for the dataset

In [514…]
```
df.describe().T
```

Out[514…]

	count	mean	std	min	25%	50%	75%	max
ID	5000.00	2500.50	1443.52	1.00	1250.75	2500.50	3750.25	5000.00
Age	5000.00	45.34	11.46	23.00	35.00	45.00	55.00	67.00
Experience	5000.00	20.10	11.47	-3.00	10.00	20.00	30.00	43.00
Income	5000.00	73.77	46.03	8.00	39.00	64.00	98.00	224.00
Family	5000.00	2.40	1.15	1.00	1.00	2.00	3.00	4.00
CCAvg	5000.00	1.94	1.75	0.00	0.70	1.50	2.50	10.00
Mortgage	5000.00	56.50	101.71	0.00	0.00	0.00	101.00	635.00
Personal_Loan	5000.00	0.10	0.29	0.00	0.00	0.00	0.00	1.00
Securities_Account	5000.00	0.10	0.31	0.00	0.00	0.00	0.00	1.00
CD_Account	5000.00	0.06	0.24	0.00	0.00	0.00	0.00	1.00
Online	5000.00	0.60	0.49	0.00	0.00	1.00	1.00	1.00
CreditCard	5000.00	0.29	0.46	0.00	0.00	0.00	1.00	1.00

In [514…]
```
df.describe(include=['category','bool'])
```

Out[514…]

	ZIPCode	Education
count	5000	5000
unique	467	3
top	94720	1
freq	169	2096

look at different levels in categorical variables

In [515…]
```
df.Education.unique()
```

Out[515…]
```
[1, 2, 3]
Categories (3, int64): [1, 2, 3]
```

- From the Data Dictionary see the following:

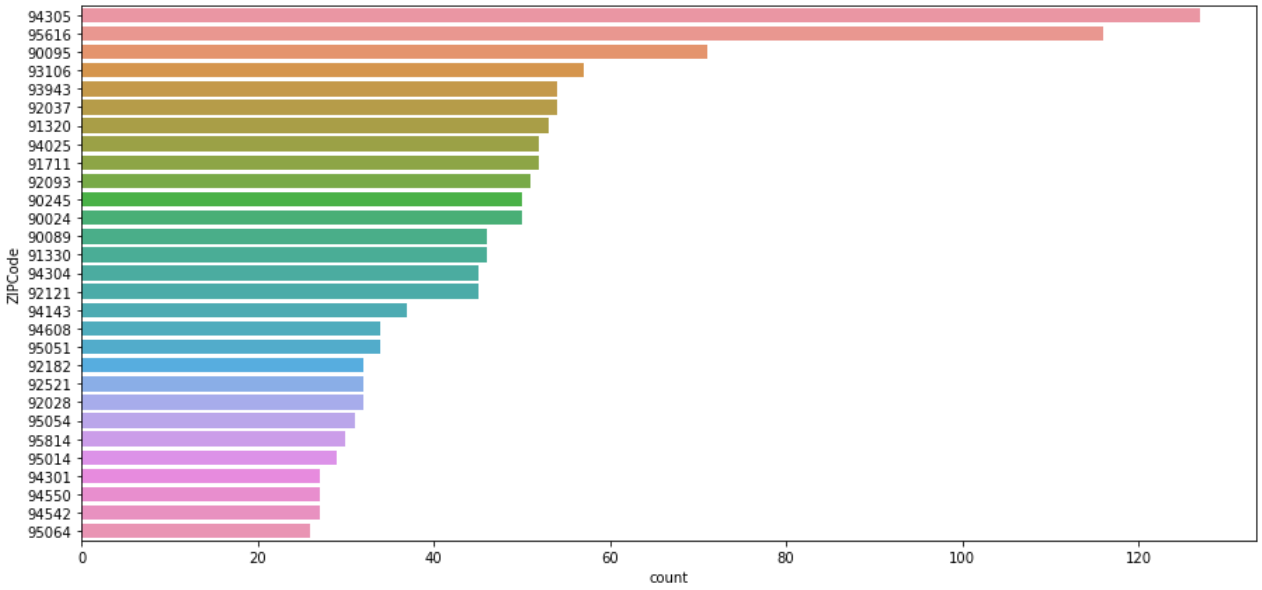
- 1: Undergrad
- 2: Graduate
- 3: Advanced/Professional

In [515…
```
df.ZIPCode.unique()
```

Out[515…
```
[91107, 90089, 94720, 94112, 91330, ..., 90068, 94970, 90813, 94404, 94598]
Length: 467
Categories (467, int64): [91107, 90089, 94720, 94112, ..., 94970, 90813, 94404, 94598]
```

In [515…
```
plt.figure(figsize=(15, 7))
sns.countplot(y="ZIPCode", data=df, order=df["ZIPCode"].value_counts().index[1:30])
```

Out[515…
```
<AxesSubplot:xlabel='count', ylabel='ZIPCode'>
```



- There are 467 unique ZIP Code values. We will have to treat this and group them by Zones. One option would be to evaluated the ZipCode Zones by the first 2 digits of the ZipCode. This will group them in geographical zones.

Ref: https://en.wikipedia.org/wiki/ZIP_Code

- 0 = Connecticut (CT), Massachusetts (MA), Maine (ME), New Hampshire (NH), New Jersey (NJ), New York (NY, Fishers Island only), Puerto Rico (PR), Rhode Island (RI), Vermont (VT), Virgin Islands (VI), Army Post Office Europe, Central Asia, and the Middle East (APO AE); Fleet Post Office Europe and the Middle East (FPO AE)
- 1 = Delaware (DE), New York (NY), Pennsylvania (PA)
- 2 = District of Columbia (DC), Maryland (MD), North Carolina (NC), South Carolina (SC), Virginia (VA), West Virginia (WV)
- 3 = Alabama (AL), Florida (FL), Georgia (GA), Mississippi (MS), Tennessee (TN), Army Post Office Americas (APO AA), Fleet Post Office Americas (FPO AA)
- 4 = Indiana (IN), Kentucky (KY), Michigan (MI), Ohio (OH)
- 5 = Iowa (IA), Minnesota (MN), Montana (MT), North Dakota (ND), South Dakota (SD), Wisconsin (WI)
- 6 = Illinois (IL), Kansas (KS), Missouri (MO), Nebraska (NE)

- 7 = Arkansas (AR), Louisiana (LA), Oklahoma (OK), Texas (TX)
- 8 = Arizona (AZ), Colorado (CO), Idaho (ID), New Mexico (NM), Nevada (NV), Utah (UT), Wyoming (WY)
- 9 = Alaska (AK), American Samoa (AS), California (CA), Guam (GU), Hawaii (HI), Marshall Islands (MH), Federated States of Micronesia (FM), Northern Mariana Islands (MP), Oregon (OR), Palau (PW), Washington (WA), Army Post Office Pacific (APO AP), Fleet Post Office Pacific (FPO AP)

In [515…
```
def zipcode_zoning(zipcode):
        return str(zipcode)[0:2]
```

In [515…
```
df['ZIPCodeZone'] = df['ZIPCode'].apply(zipcode_zoning)

df.ZIPCodeZone.unique()
```

Out[515…
```
array(['91', '90', '94', '92', '93', '95', '96'], dtype=object)
```

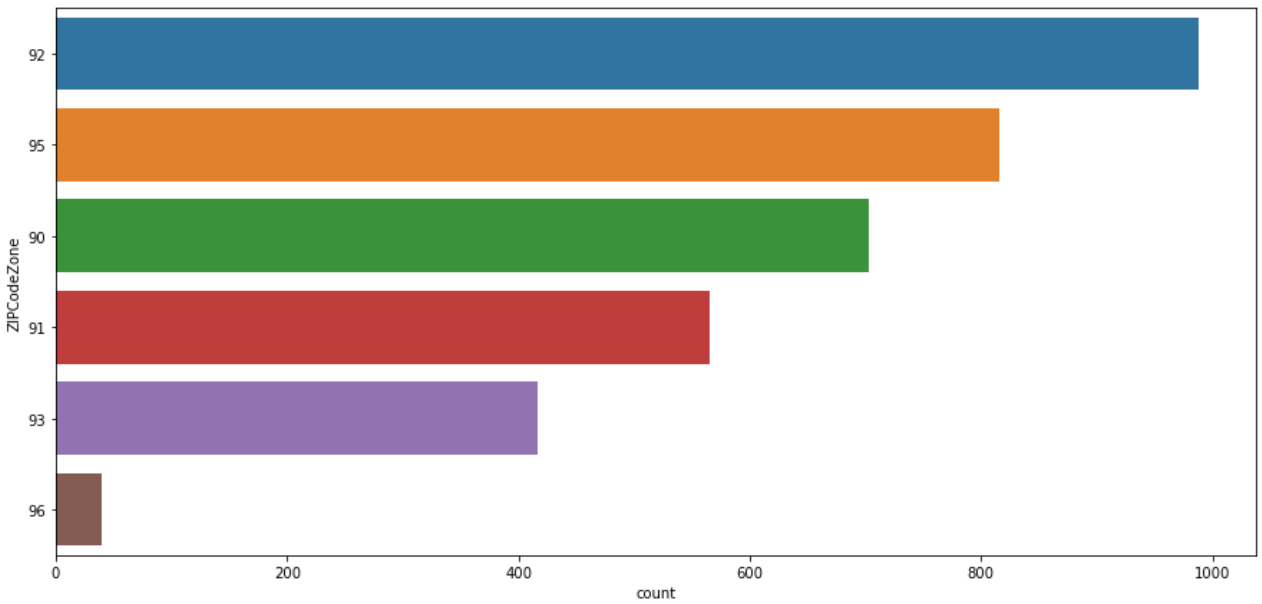
In [515…
```
df.head()
```

Out[515…

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_Loan	Securi
0	1	25	1	49	91107	4	1.60	1	0	0	
1	2	45	19	34	90089	3	1.50	1	0	0	
2	3	39	15	11	94720	1	1.00	1	0	0	
3	4	35	9	100	94112	1	2.70	2	0	0	
4	5	35	8	45	91330	4	1.00	2	0	0	

In [515…
```
plt.figure(figsize=(15, 7))
sns.countplot(y="ZIPCodeZone", data=df, order=df["ZIPCodeZone"].value_counts().index[1:
```

Out[515…
```
<AxesSubplot:xlabel='count', ylabel='ZIPCodeZone'>
```



Exploratory Data Analysis

Univariate Analysis

In [515…

```
def histogram_boxplot(feature , figsize=(15,10) , bins=None):
    """ Histogram and Boxplot combined
    feature: 1-d feature array
    figsize: size of figg.default (15,10)
    bins: number of bins.default None/auto
    """
    mean = feature.mean()
    median = feature.median()
    mode = feature.mode()

    f2, (ax_box2 , ax_hist2) = plt.subplots(nrows = 2, # num of rows of the subplot. gr
                                            sharex = True, # x-axis will be shared amon
                                            gridspec_kw = { "height_ratios": (.25 , .75
                                            figsize = figsize
                                           ) # create the 2 subplots

    sns.boxplot(feature , ax = ax_box2 , showmeans = True , color = 'red') # boxplot wi
    if bins:
        sns.distplot(feature , kde = True , ax = ax_hist2, bins = bins)
    else:
        sns.distplot( feature , kde = True , ax = ax_hist2 )
    ax_hist2.axvline( mean , color = 'green' , linestyle='-' , linewidth = 3 , label =
    ax_hist2.axvline( median , color = 'yellow' , linestyle='-' , linewidth = 6 , label
    ax_hist2.axvline( mode[0] , color = 'black' , linestyle='-' , label = 'mode' ) # ad
    ax_hist2.legend()

    print( 'Mean:'+ str( mean ) )
    print( 'Median:'+ str( median ) )
    print( 'Mode:'+ str( mode[0] ) )
```

In [515…

```
def bar_count_pct( feature , figsize=(10,7) ):
    """
    feature : 1-d categorical feature array
    """
    mode = feature.mode()
    freq = feature.value_counts().max()

    #if isinstance(feature , int):
    #    cnt = feature.unique()
    #else:
    #    cnt  = feature.unique().value_counts().sum()

    plt.figure(figsize=figsize)

    ax = sns.countplot(feature)

    total = len(feature) # length of the column
    for p in ax.patches:
        percentage = '{:.1f}%'.format( 100 * p.get_height() / total ) # percentage of e
        x = p.get_x() + p.get_width() / 2 - 0.05 # width of the plot
        y = p.get_y() + p.get_height() # height of the plot
        ax.annotate( percentage , (x,y), size = 12) # annotate the percentage

    print( 'Top:'+ str( mode[0] ) )
```

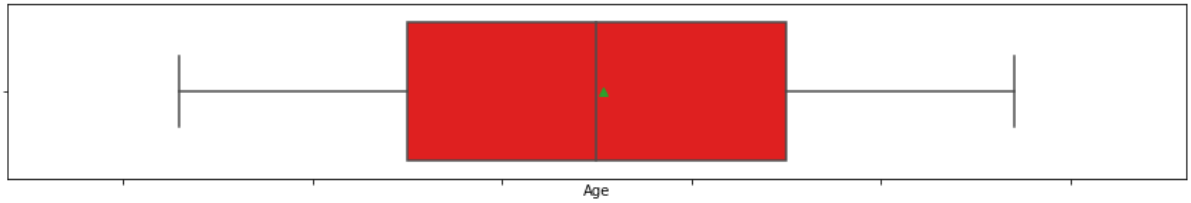
```
    print( 'Freq:'+ str( freq ) )
    #print( 'Uniq:'+ str( cnt ) )
```

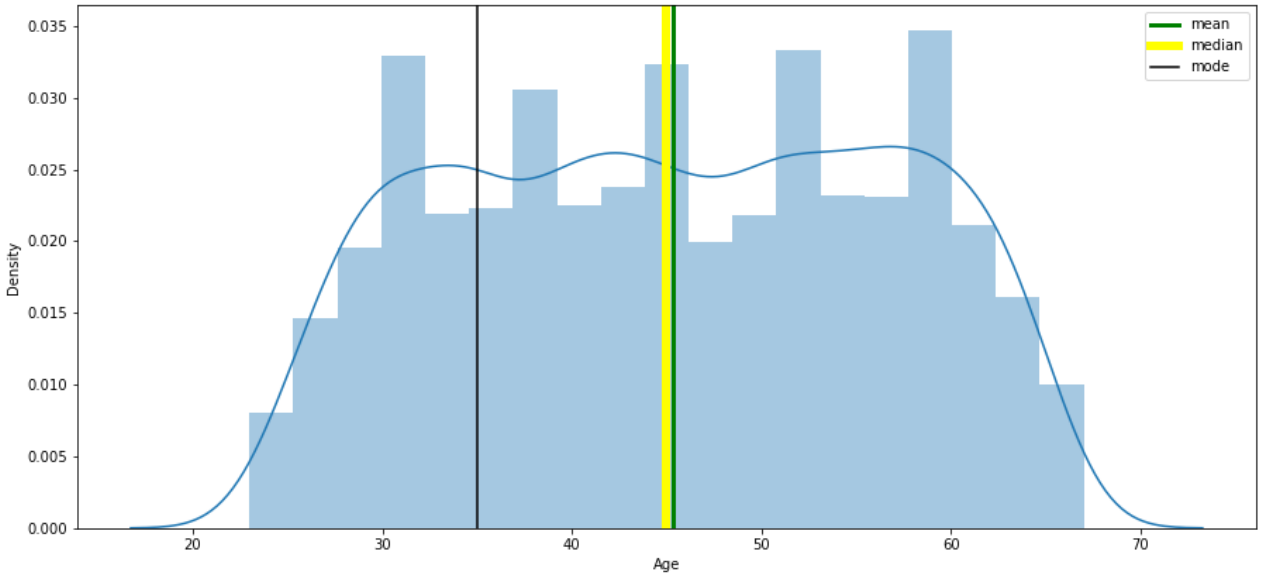
Observations on Age

In [515…

```
histogram_boxplot(df.Age)
```

```
Mean:45.3384
Median:45.0
Mode:35
```





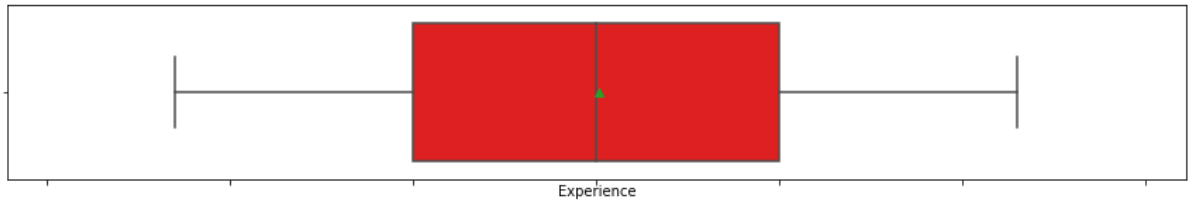
- Age feature is slightly right-skewed, but overall the distribution looks normal.
- There are no outliers for this feature.

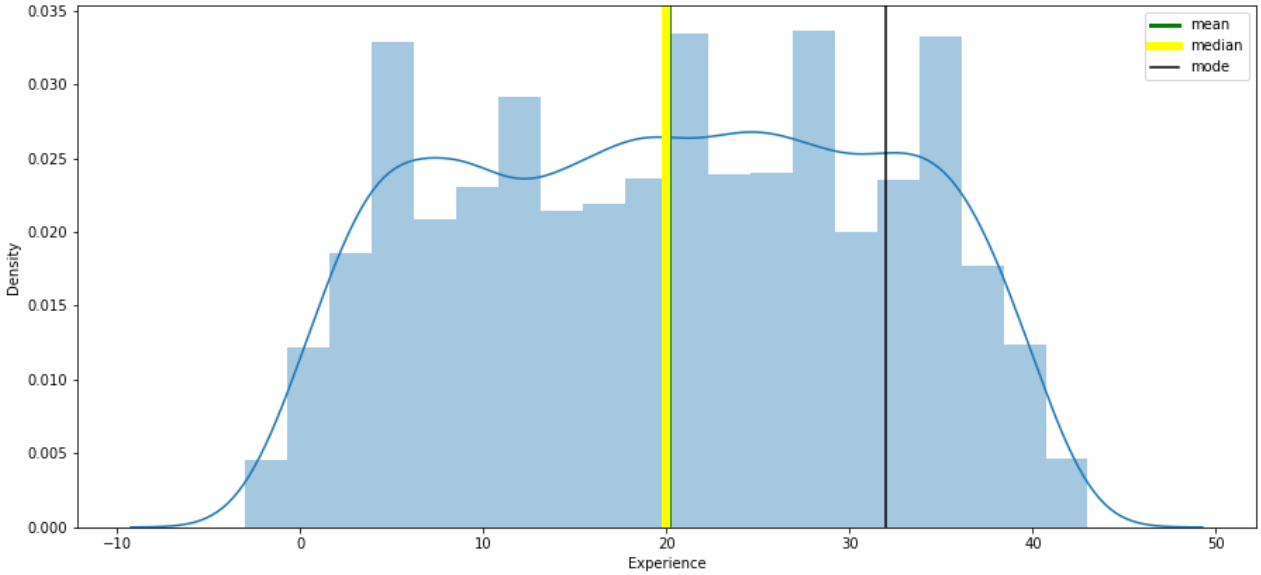
Observations on Experience

In [516…

```
histogram_boxplot(df.Experience)
```

```
Mean:20.1046
Median:20.0
Mode:32
```





- There seems to be NEGATIVE values for the Experience variable. We can assume this is an error since Experience should be values 0 or greater. We will have to convert any negative values to zero.

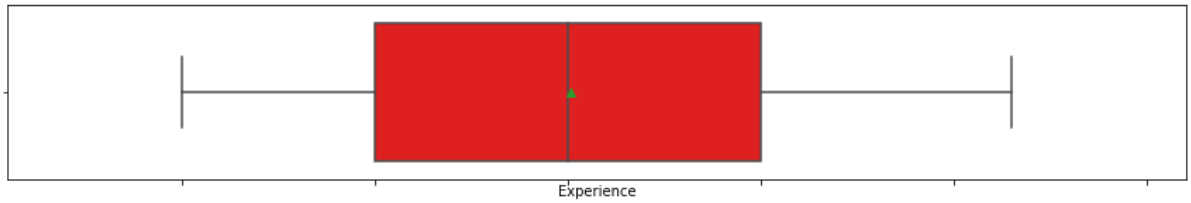
In [516…
```
def convertNegativeExperienceToZero(experience):
    if experience < 0 :
        return 0
    else:
        return experience
```

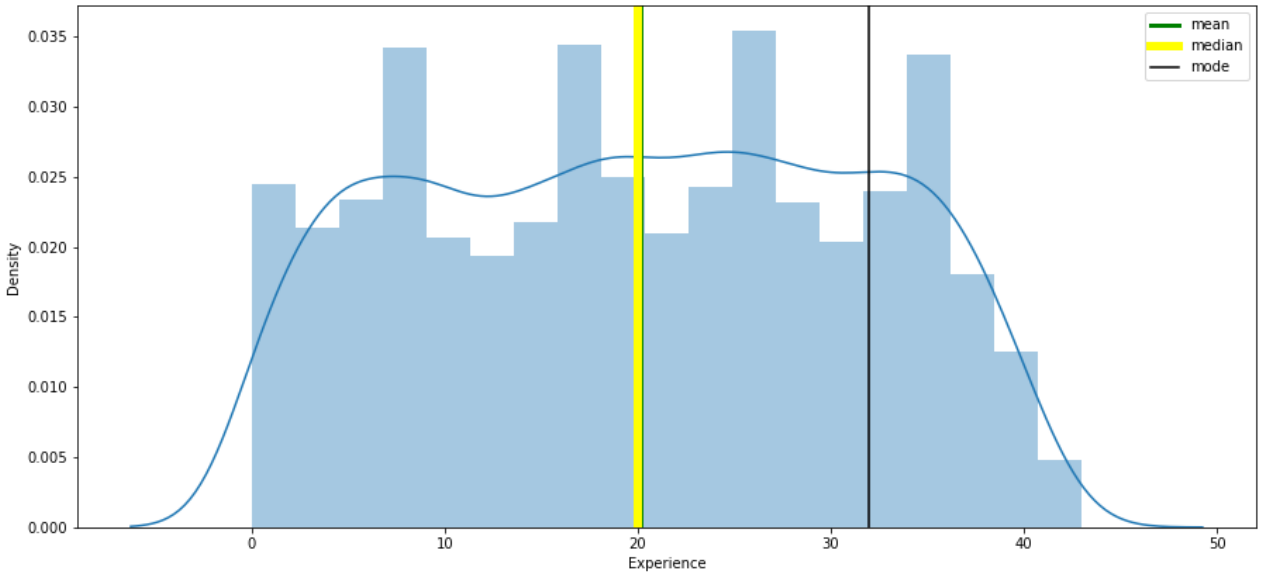
In [516…
```
df['Experience'] = df['Experience'].apply(convertNegativeExperienceToZero)
```

- Let run the Boxplot and Histogram again.

In [516…
```
histogram_boxplot(df.Experience)
```

```
Mean:20.1196
Median:20.0
Mode:32
```





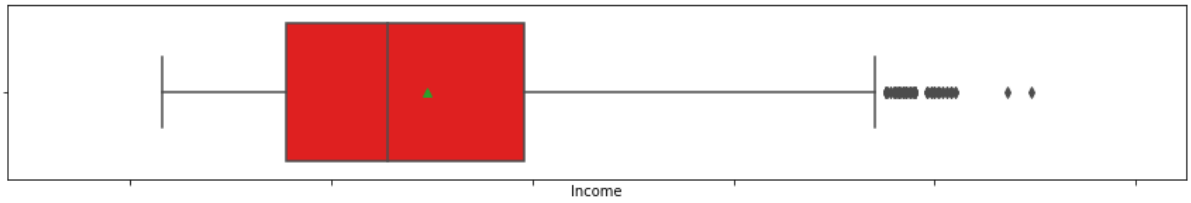
- Values with Zero experience has increased after the conversion.
- Mean value increased very little. Mean:20.1196
- Experience does not look skewed(if so, it is very slightly), but overall the distribution looks normal.
- There are no outliers for this feature.

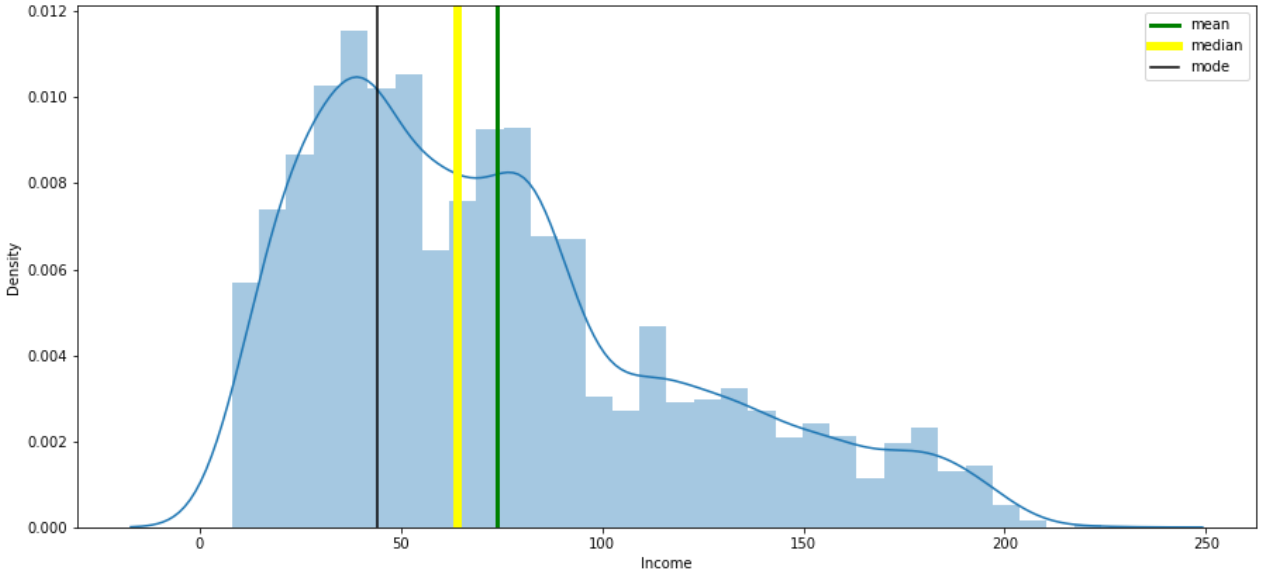
Observations on Income

In [516…

```
histogram_boxplot(df.Income)
```

```
Mean:73.7742
Median:64.0
Mode:44
```





In [516…
```
# Lets us look at quantile of Income
df.Income.quantile([.1,.2,.3,.4,.5,.6,.7,.8,.9,.95,.98,.99,1])
```

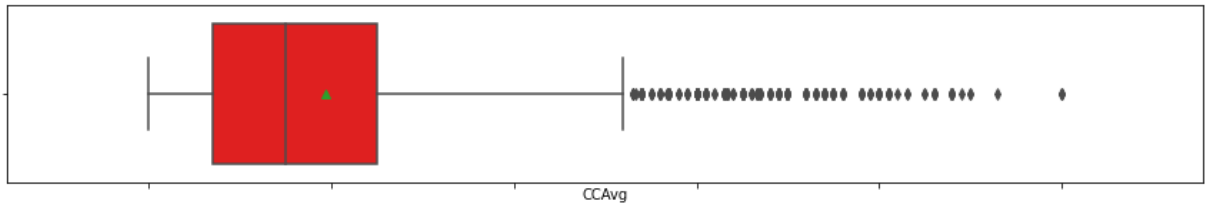
Out[516…
```
0.10    22.00
0.20    33.00
0.30    42.00
0.40    52.00
0.50    64.00
0.60    78.00
0.70    88.30
0.80   113.00
0.90   145.00
0.95   170.00
0.98   185.00
0.99   193.00
1.00   224.00
Name: Income, dtype: float64
```

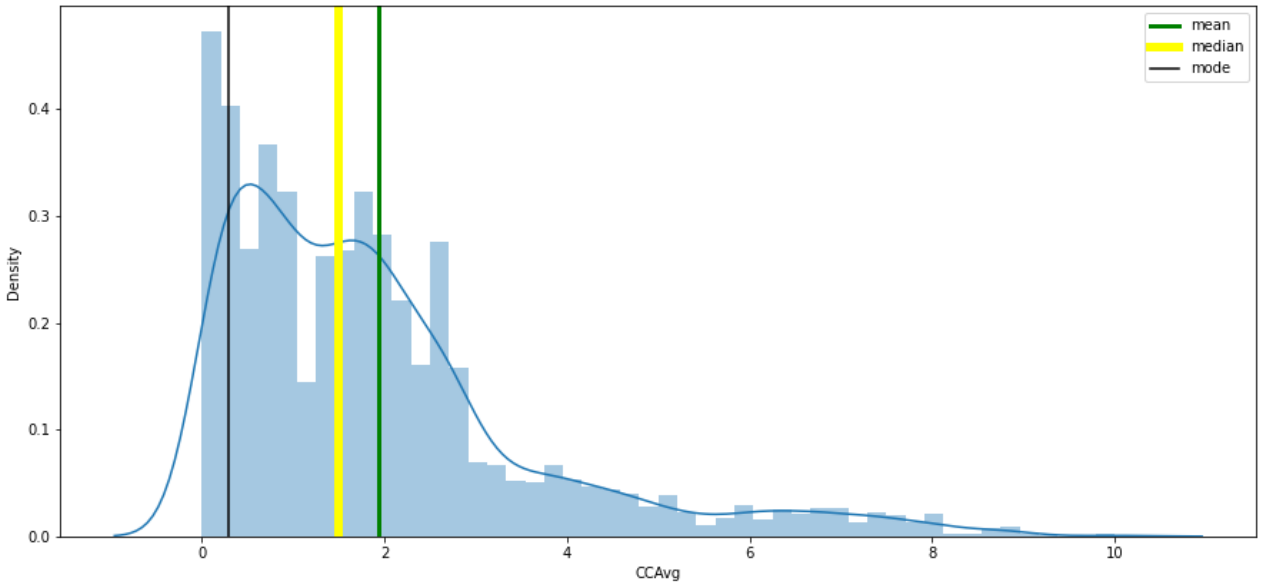
- Income is right-skewed.
- There are outliers present for this feature.

Observations on CCAvg

In [516…
```
histogram_boxplot(df.CCAvg)
```

```
Mean:1.9379380000000053
Median:1.5
Mode:0.3
```





In [516…
```
# Lets us look at quantile of CCAvg
df.CCAvg.quantile([.1,.2,.3,.4,.5,.6,.7,.8,.9,.95,.98,.99,1])
```

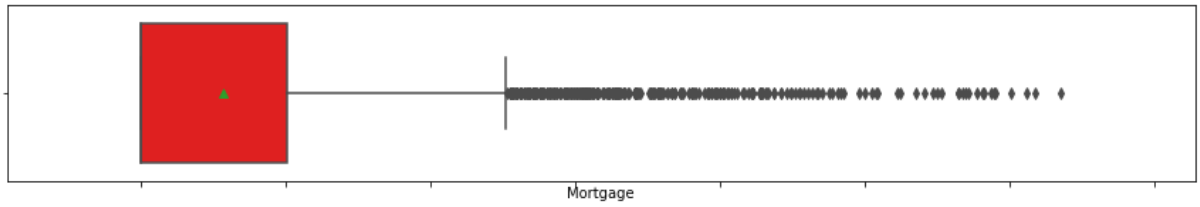
Out[516…
```
0.10    0.30
0.20    0.50
0.30    0.80
0.40    1.20
0.50    1.50
0.60    1.90
0.70    2.30
0.80    2.80
0.90    4.30
0.95    6.00
0.98    7.30
0.99    8.00
1.00   10.00
Name: CCAvg, dtype: float64
```

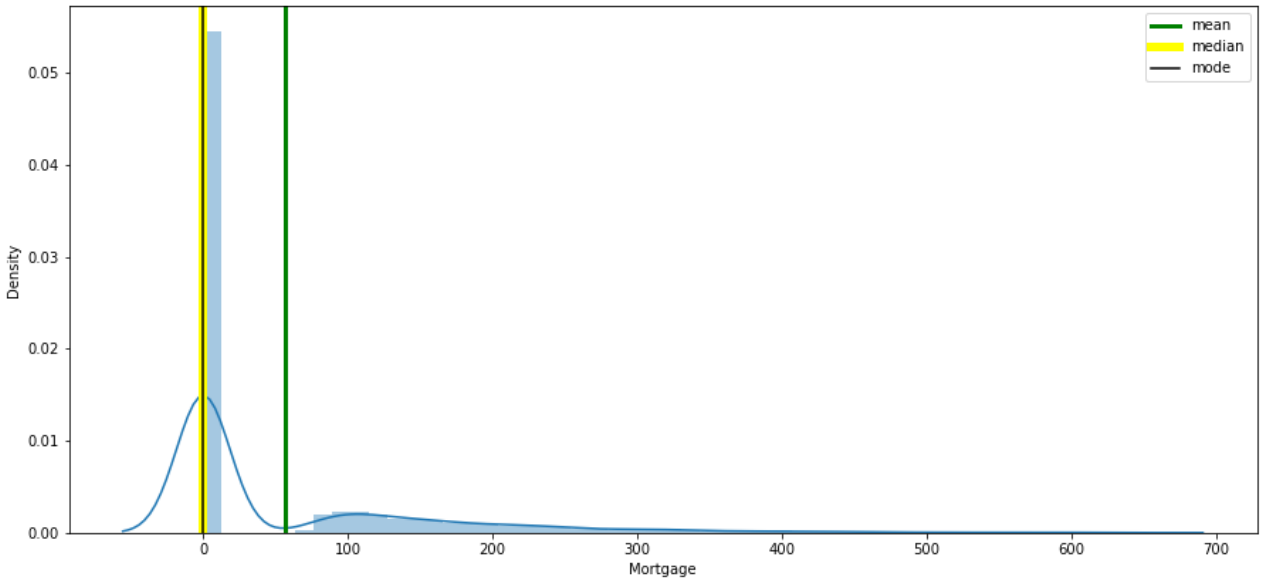
- CCAvg is highly right-skewed.
- There are outliers present for this feature.
- Majority observation has 0 CCAvg but it is right skewed because of some outliers

Observations on Mortgage

In [516…
```
histogram_boxplot(df.Mortgage)
```

```
Mean:56.4988
Median:0.0
Mode:0
```





In [516…
```
# Lets us look at quantile of Mortgage
df.Mortgage.quantile([.1,.2,.3,.4,.5,.6,.7,.8,.9,.95,.98,.99,1])
```

Out[516…
```
0.10     0.00
0.20     0.00
0.30     0.00
0.40     0.00
0.50     0.00
0.60     0.00
0.70    78.00
0.80   123.00
0.90   200.00
0.95   272.00
0.98   366.04
0.99   431.01
1.00   635.00
Name: Mortgage, dtype: float64
```

In [ ]:

In [ ]:

- Mortgage is right-skewed.
- There are many outliers present for this feature.
- Majority observation has 0 Mortgage but it is right skewed because of some outliers

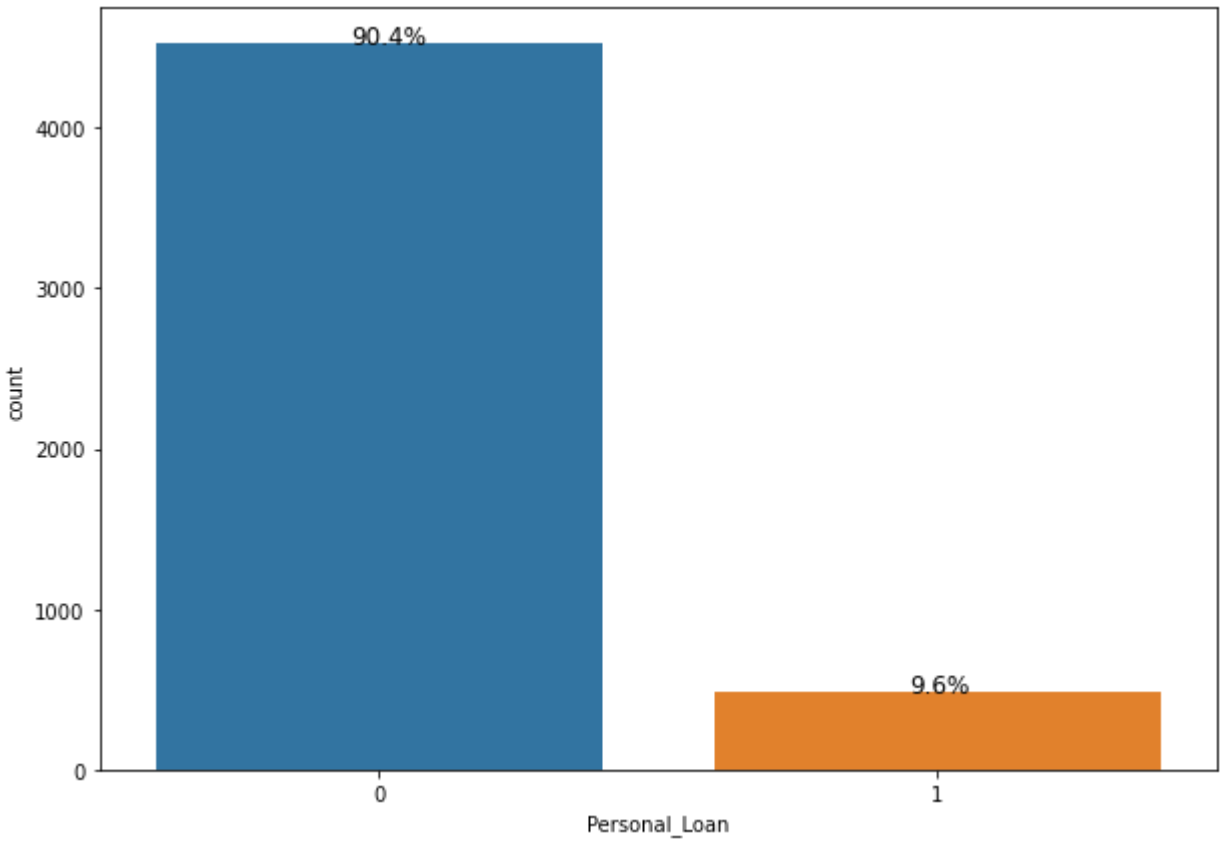
In [ ]:

Observations on Personal_Loan (Dependant Variable)

In [517…
```
print('Took Personal Loan\n' , df['Personal_Loan'].value_counts(normalize=True) , '\n')
```

```
bar_count_pct(df.Personal_Loan)
```

```
Took Personal Loan
 0   0.90
1   0.10
Name: Personal_Loan, dtype: float64

Top:0
Freq:4520
```



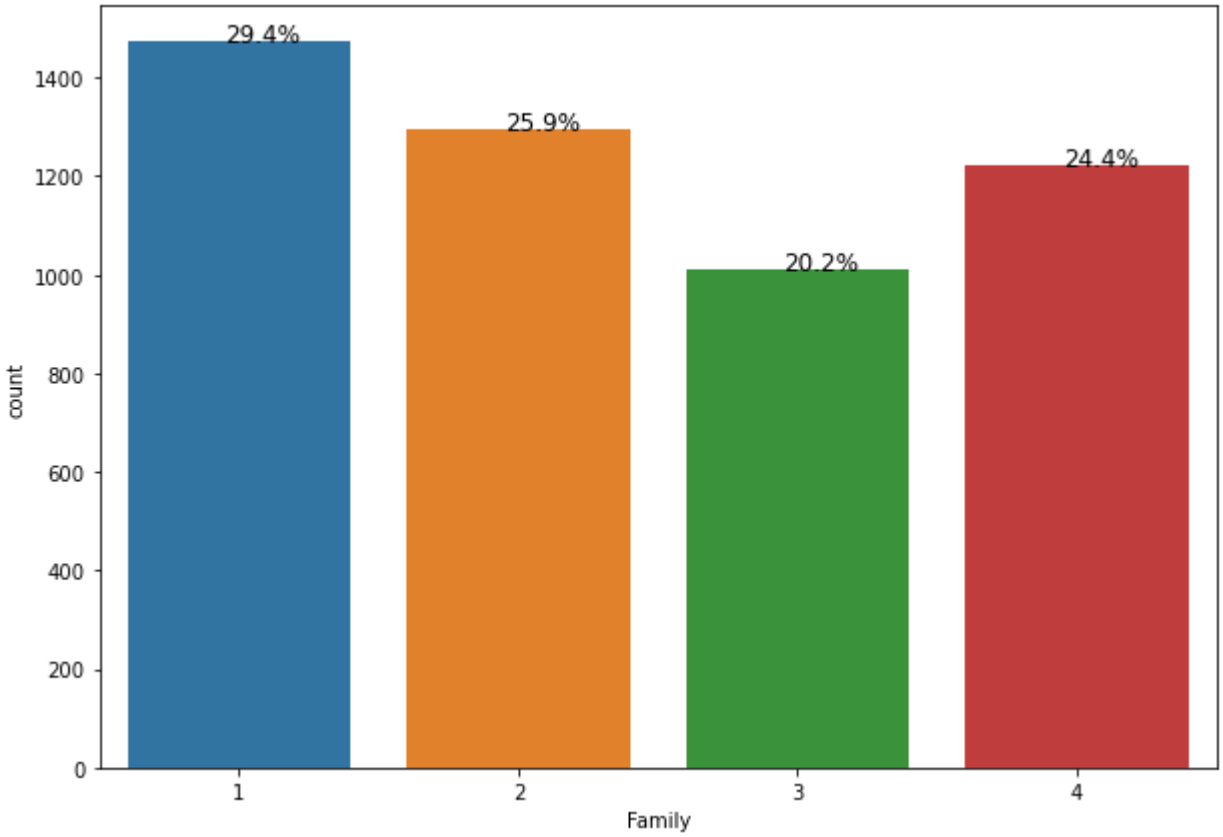
- Mode frequent Personal_Loan is False(0) with 90.4% (4520).
- Only 9.6% of Liability Customers converted.
- There are 2 (True/False or 0/1) unique values.

Observations on Family

In [517…

```
bar_count_pct(df.Family)
```

```
Top:1
Freq:1472
```



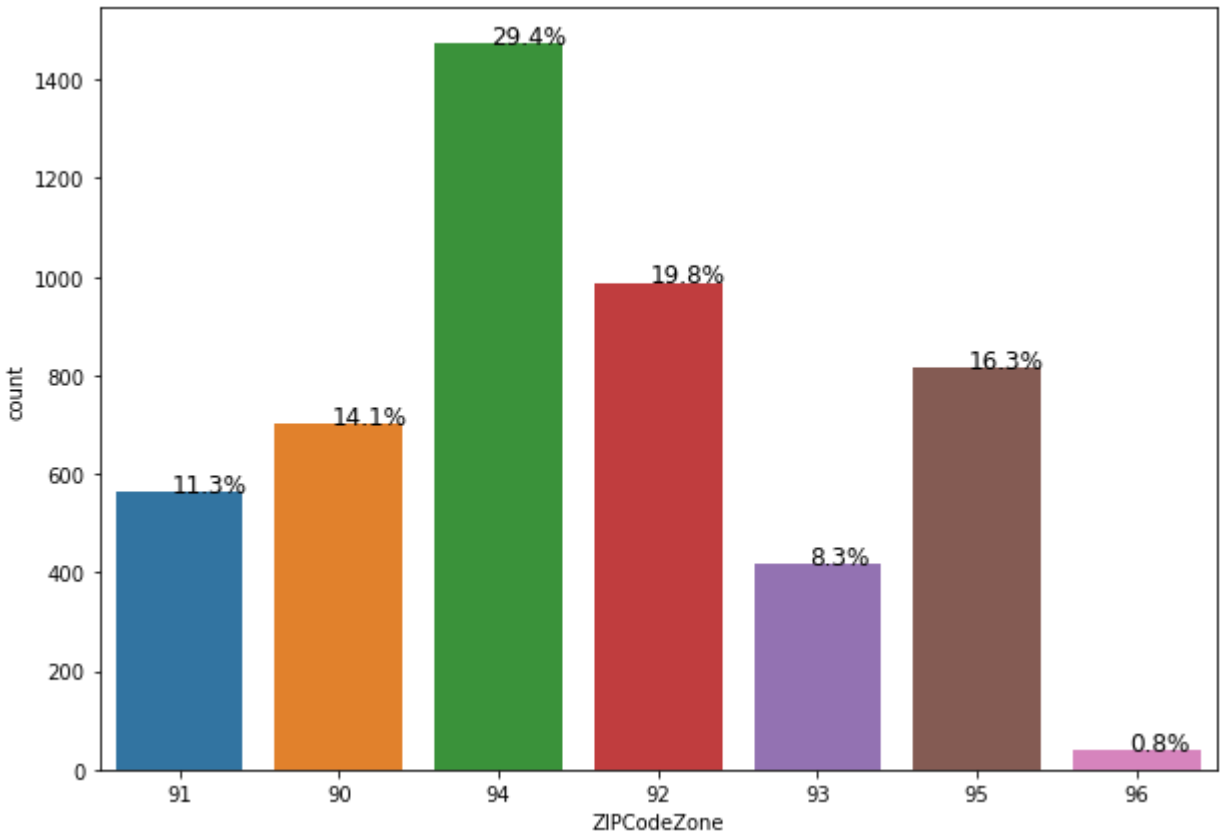
- Mode frequent `Family` size is 1 is 94 with 29.4% (1472).
- There are 4 unique values.

Observations on ZipCodeZone

In [517…

```
bar_count_pct(df.ZIPCodeZone)
```

```
Top:94
Freq:1472
```



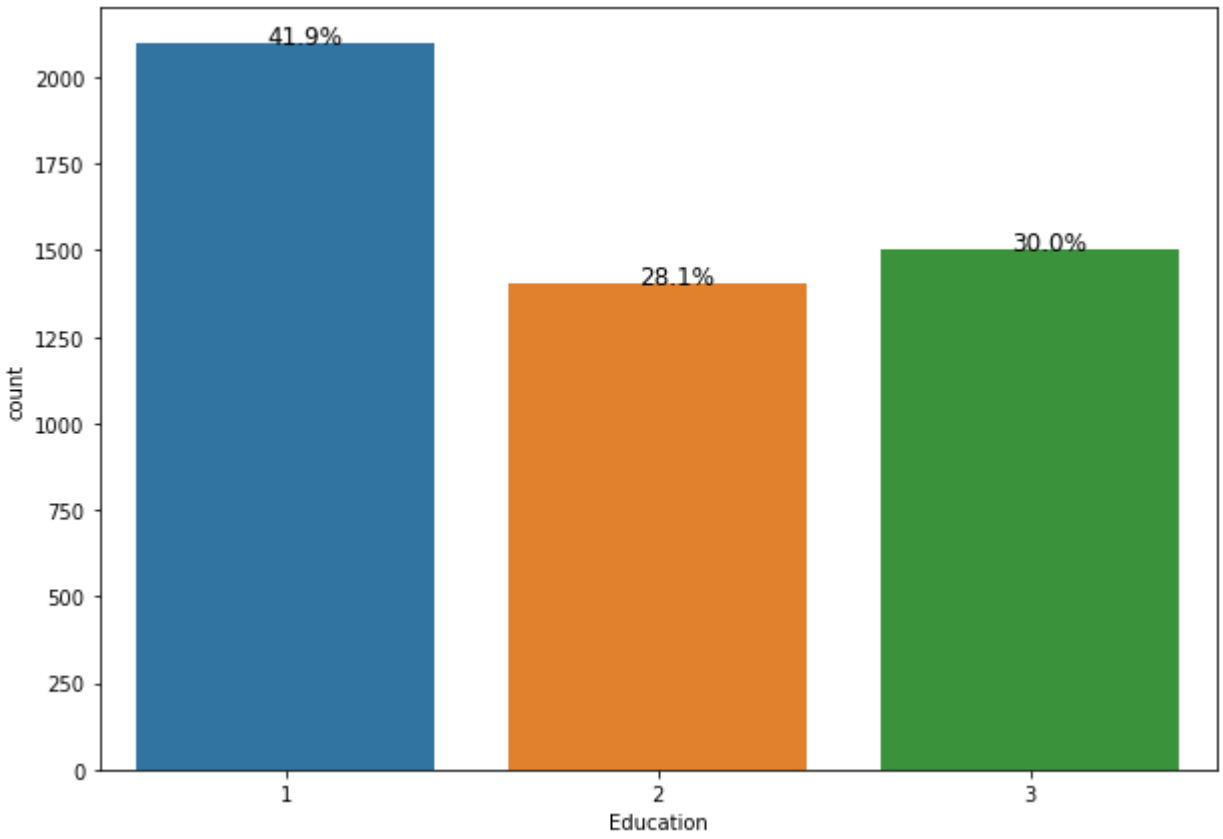
- Mode frequent ZipCode Zone is 94 with 29.4% (1472).
- There are 7 unique values.

Observations on Education

In [517…

```
bar_count_pct(df.Education)
```

```
Top:1
Freq:2096
```



- Mode frequent Education level 1 with 41.9% (2096).
- There are 3 unique values.

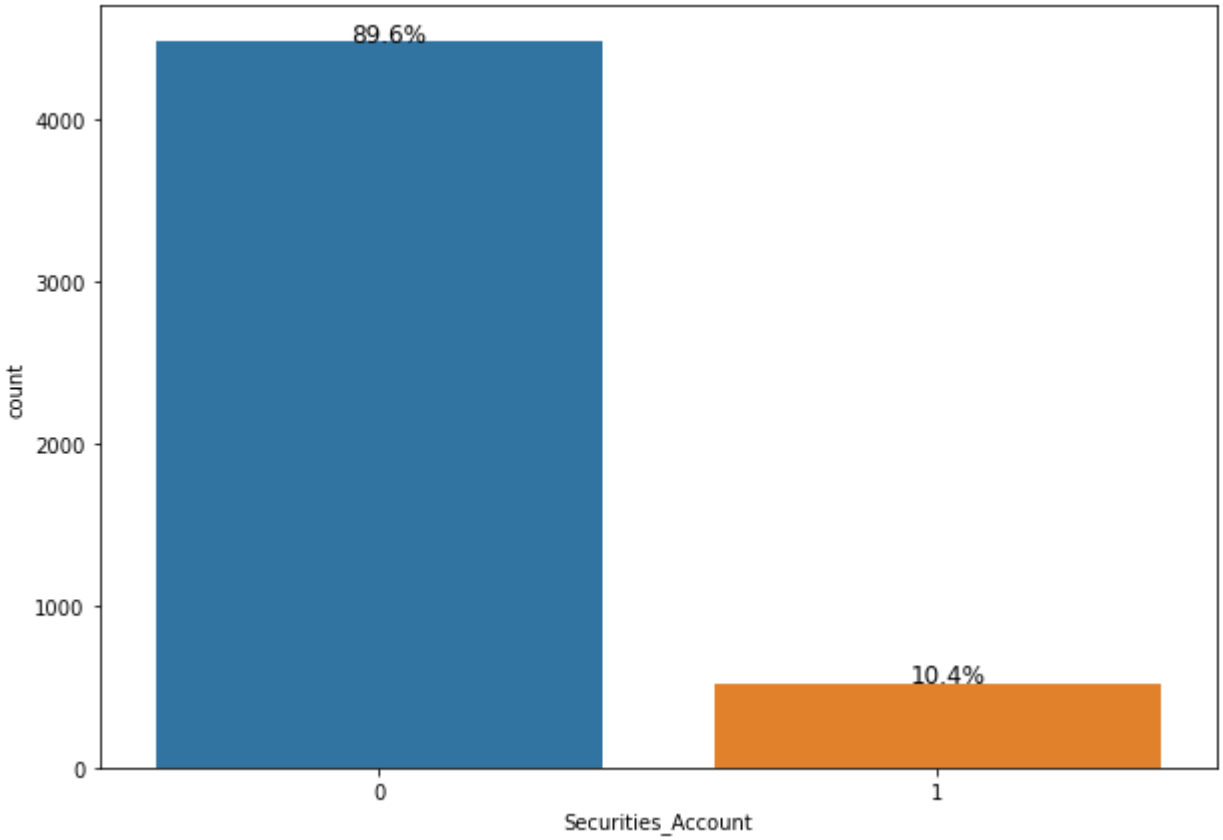
9 Personal_Loan 5000 non-null int64 10 Securities_Account 5000 non-null int64 11 CD_Account 5000 non-null int64 12 Online 5000 non-null int64 13 CreditCard 5000 non-null int64

Observations on Securities_Account

In [517…

```
bar_count_pct(df.Securities_Account)
```

```
Top:0
Freq:4478
```



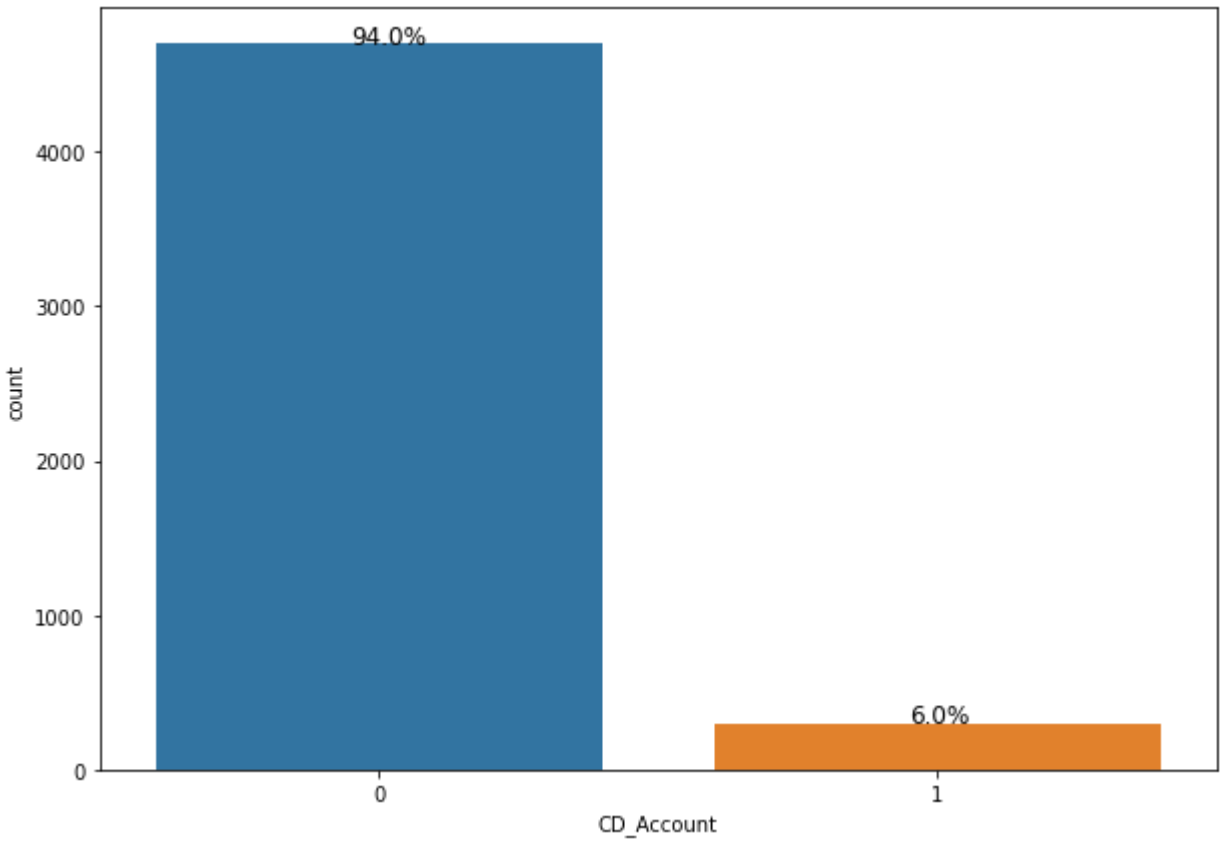
- Mode frequent Securities_Account is False(0) with 89.6% (4478).
- There are 2 (True/False or 0/1) unique values.

Observations on CD_Account

In [517…

```
bar_count_pct(df.CD_Account)
```

```
Top:0
Freq:4698
```



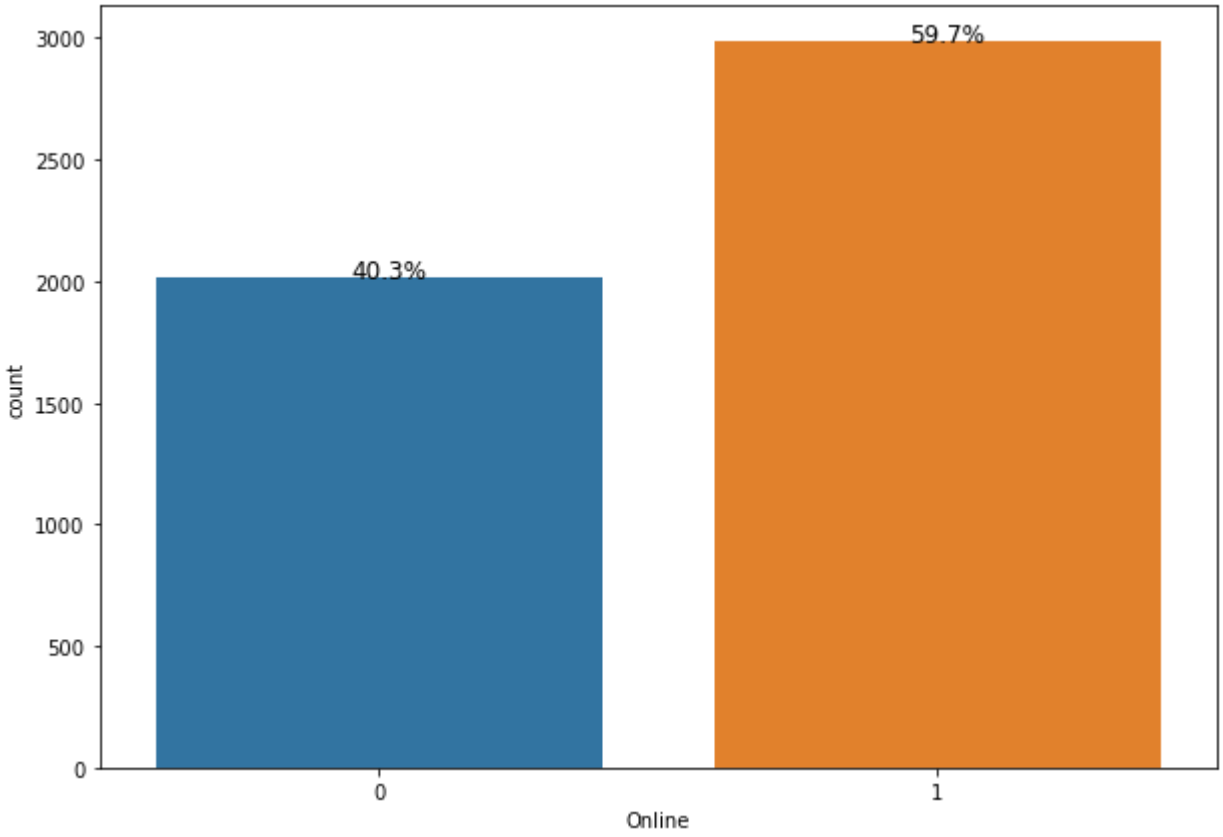
- Mode frequent CD_Account is False(0) with 94% (4698).
- There are 2 (True/False or 0/1) unique values.

Observations on Online

In [517…

```
bar_count_pct(df.Online)
```

```
Top:1
Freq:2984
```



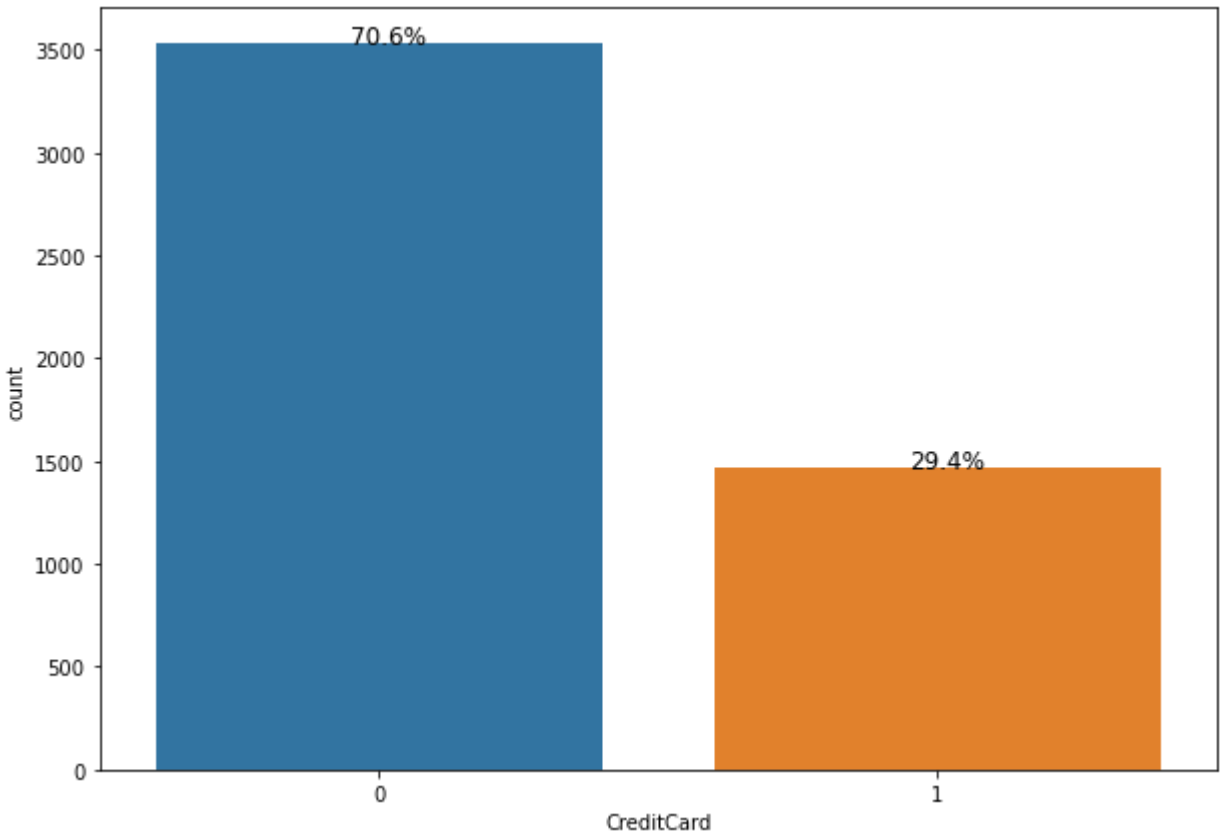
- Mode frequent Online value is True(1) with 59.7% (2984).
- There are 2 (True/False or 0/1) unique values.

Observations on CreditCard

In [517…

```
bar_count_pct(df.CreditCard)
```

```
Top:0
Freq:3530
```



- Mode frequent CreditCard value is False(0) with 70.6% (3530).
- There are 2 (True/False or 0/1) unique values.

In [ ]:

Bivariate Analysis

0 ID 5000 non-null int64
1 Age 5000 non-null int64
2 Experience 5000 non-null int64
3 Income 5000 non-null int64
4 ZIPCode 5000 non-null category 5 Family 5000 non-null int64
6 CCAvg 5000 non-null float64 7 Education 5000 non-null category 8 Mortgage 5000 non-null int64
9 Personal_Loan 5000 non-null int64
10 Securities_Account 5000 non-null int64
11 CD_Account 5000 non-null int64
12 Online 5000 non-null int64
13 CreditCard 5000 non-null int64

In [517…

```
### Age vs Personal_Loan
```

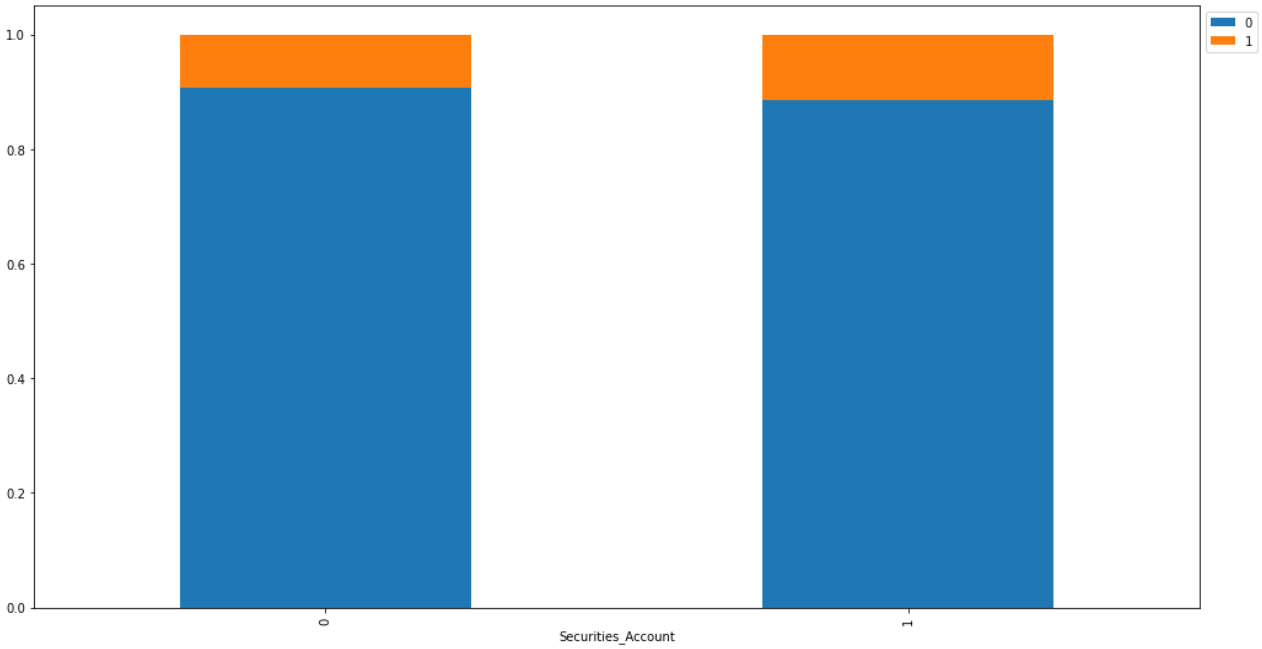
Securities_Account vs Personal_Loan

In [517…

```
tab1 = pd.crosstab(df.Securities_Account , df.Personal_Loan , margins=True)
print(tab1)
print('-'*120)
```

```
tab = pd.crosstab(df.Securities_Account , df.Personal_Loan , normalize='index')
tab.plot(kind='bar',stacked=True,figsize=(17,9))
plt.legend(loc="upper left", bbox_to_anchor=(1,1));
```

```
Personal_Loan          0    1   All
Securities_Account
0                   4058  420  4478
1                    462   60   522
All                 4520  480  5000
--------------------------------------------------------------------------------------------
--------------------------------
```



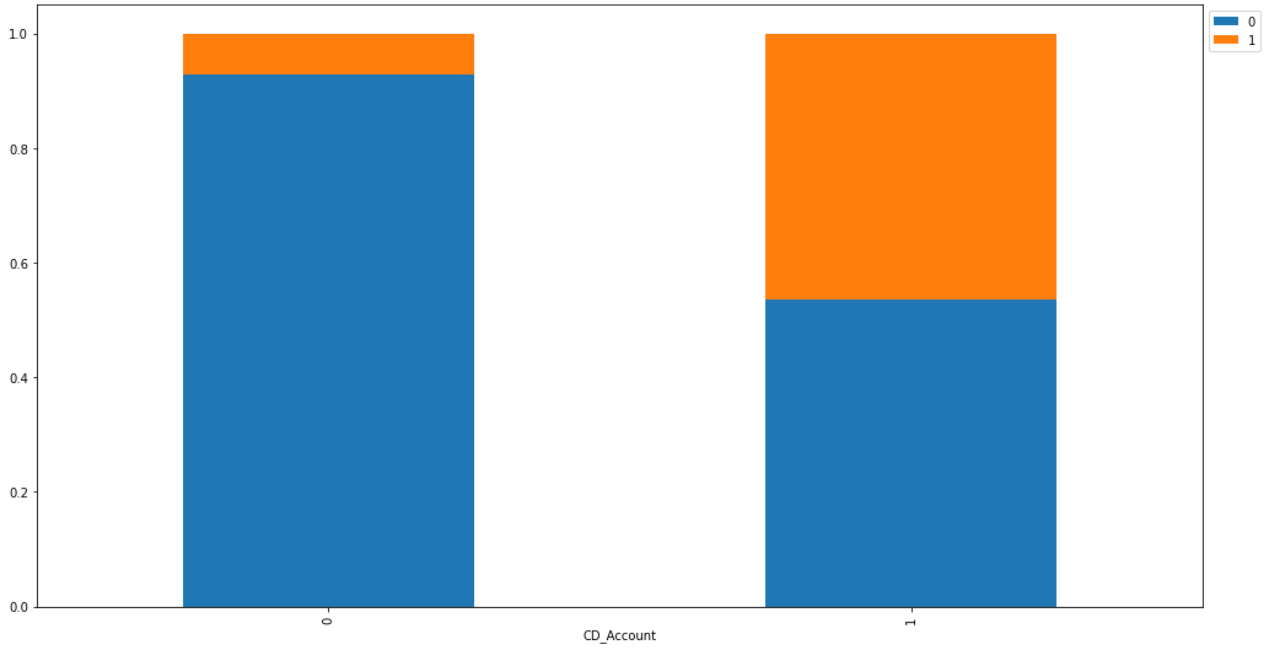
- Proportionally,there is no significant difference between customer that have `Securities_Account` vs those who do not. In both cases about 10% of customers converted to Asset Customers.

CD_Account vs Personal_Loan

In [518…

```
tab1 = pd.crosstab(df.CD_Account , df.Personal_Loan , margins=True)
print(tab1)
print('-'*120)
tab = pd.crosstab(df.CD_Account , df.Personal_Loan , normalize='index')
tab.plot(kind='bar',stacked=True,figsize=(17,9))
plt.legend(loc="upper left", bbox_to_anchor=(1,1));
```

```
Personal_Loan     0    1   All
CD_Account
0              4358  340  4698
1               162  140   302
All            4520  480  5000
--------------------------------------------------------------------------------------------
--------------------------------
```



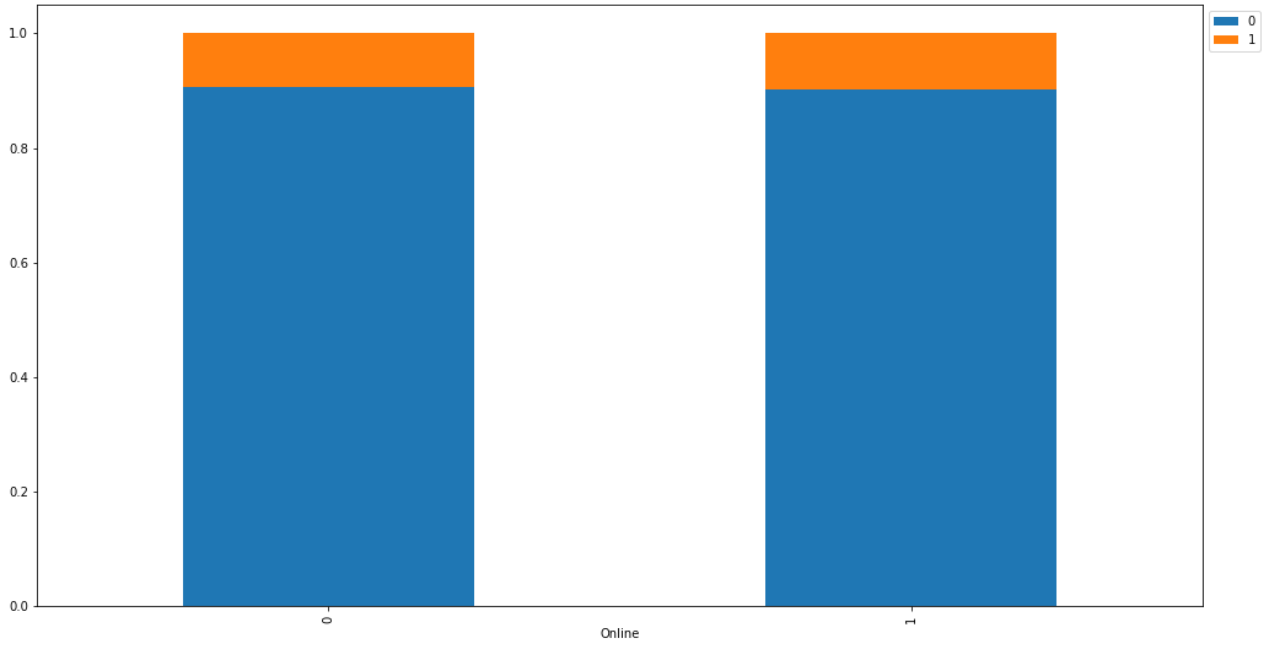
Customers that cas `CD_Account` have converted at a 46% rate, versus 7% of customer that do not have a `CD_ACCOUNT` .

Online vs Personal_Loan

In [518…

```
tab1 = pd.crosstab(df.Online , df.Personal_Loan , margins=True)
print(tab1)
print('-'*120)
tab = pd.crosstab(df.Online , df.Personal_Loan , normalize='index')
tab.plot(kind='bar',stacked=True,figsize=(17,9))
plt.legend(loc="upper left", bbox_to_anchor=(1,1));
```

```
Personal_Loan     0    1   All
Online
0              1827  189  2016
1              2693  291  2984
All            4520  480  5000
------------------------------------------------------------------------------------------
--------------------------------
```



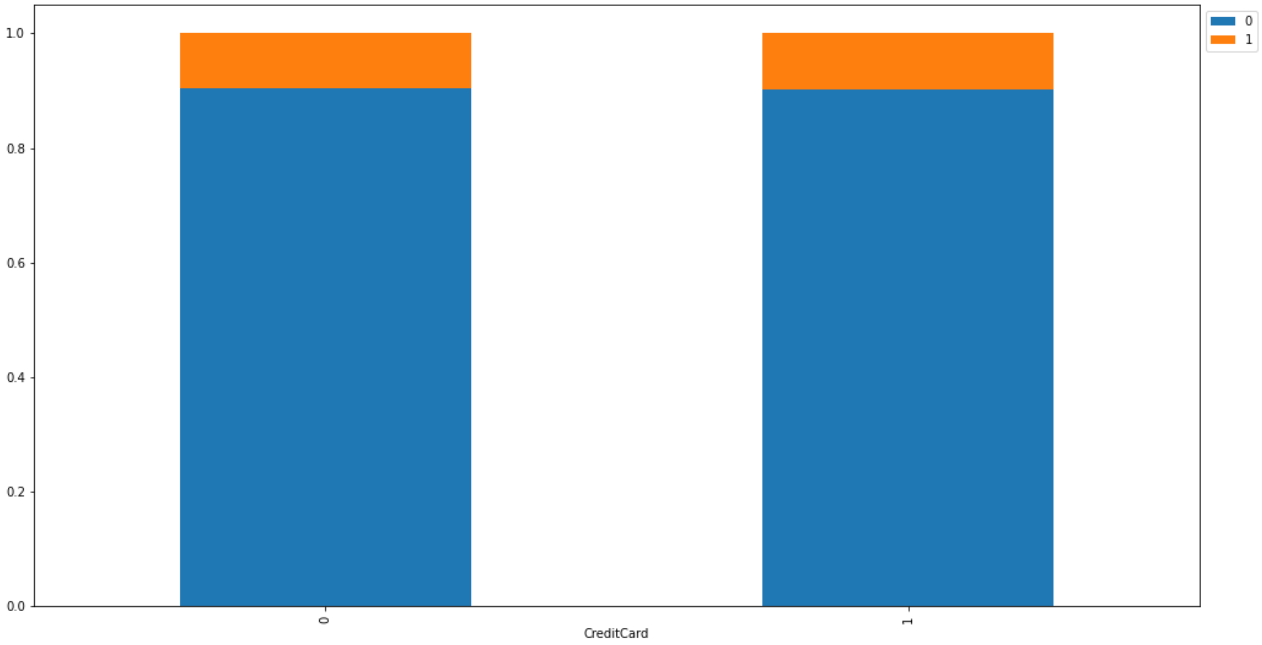
- Proportionally,there is no significant difference between customer that have `Online` vs those who do not. In both cases about 10% of customers converted to Asset Customers.

CreditCard vs Personal_Loan

In [518…

```
tab1 = pd.crosstab(df.CreditCard , df.Personal_Loan , margins=True)
print(tab1)
print('-'*120)
tab = pd.crosstab(df.CreditCard , df.Personal_Loan , normalize='index')
tab.plot(kind='bar',stacked=True,figsize=(17,9))
plt.legend(loc="upper left", bbox_to_anchor=(1,1));
```

```
Personal_Loan     0    1   All
CreditCard
0              3193  337  3530
1              1327  143  1470
All            4520  480  5000
------------------------------------------------------------------------------------------
--------------------------------
```



- Proportionally,there is no significant difference between customer that have `CreditCard` vs those who do not. In both cases about 10% of customers converted to Asset Customers.

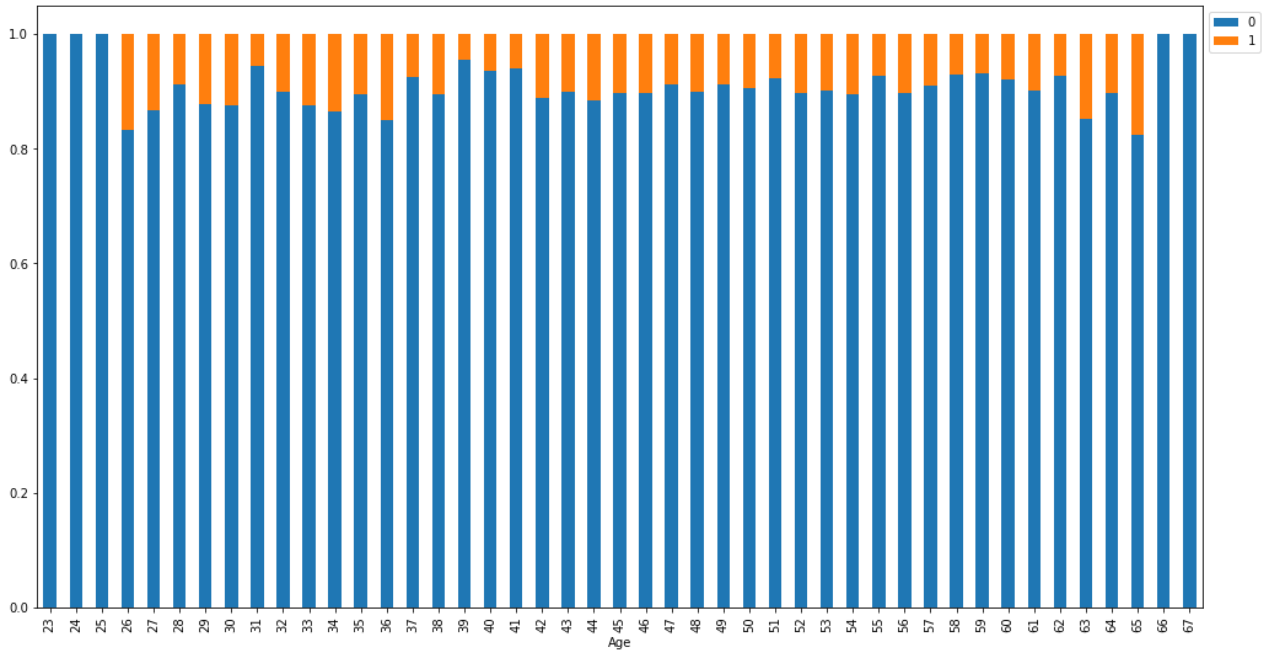
In [ ]:

In [518…

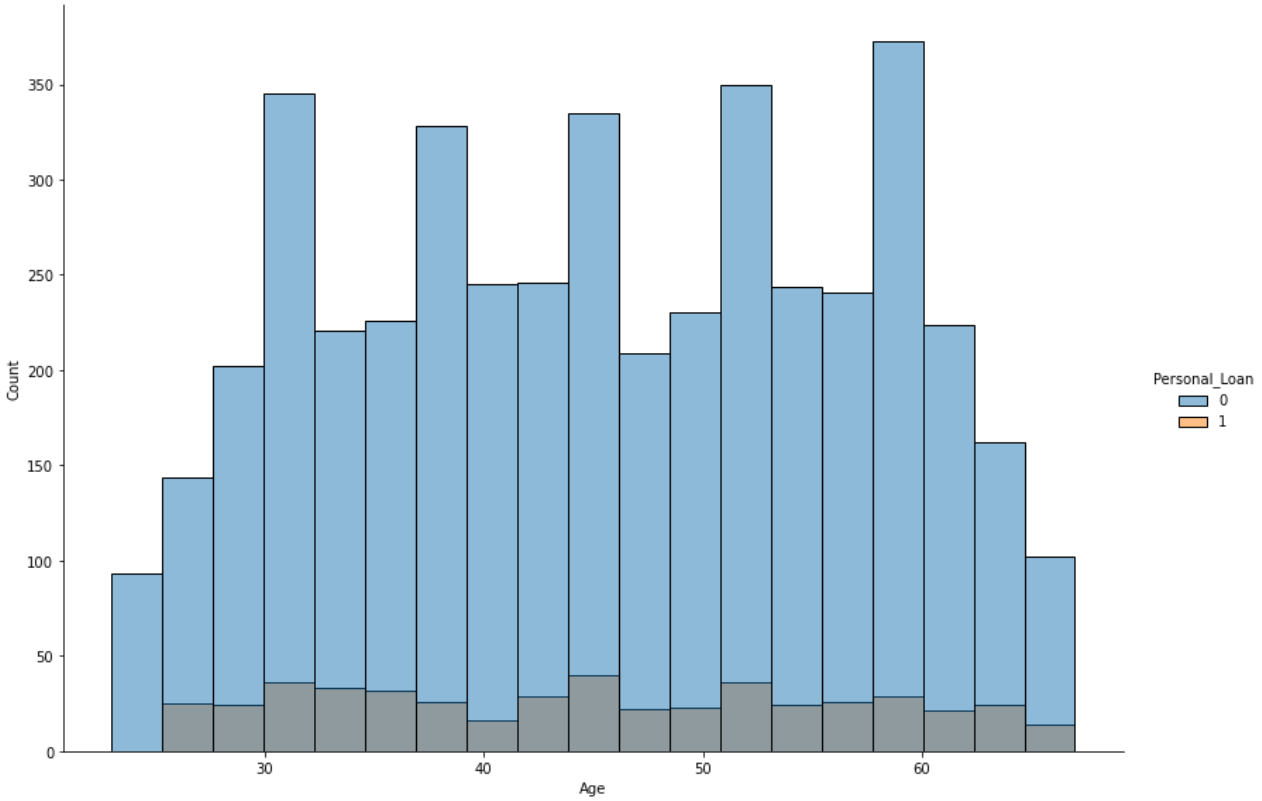
```
tab1 = pd.crosstab(df.Age , df.Personal_Loan , margins=True)
print(tab1)
print('-'*120)
tab = pd.crosstab(df.Age , df.Personal_Loan , normalize='index')
tab.plot(kind='bar',stacked=True,figsize=(17,9))
plt.legend(loc="upper left", bbox_to_anchor=(1,1));
```

```
Personal_Loan     0    1   All
Age
23               12    0    12
24               28    0    28
25               53    0    53
```

26	65	13	78
27	79	12	91
28	94	9	103
29	108	15	123
30	119	17	136
31	118	7	125
32	108	12	120
33	105	15	120
34	116	18	134
35	135	16	151
36	91	16	107
37	98	8	106
38	103	12	115
39	127	6	133
40	117	8	125
41	128	8	136
42	112	14	126
43	134	15	149
44	107	14	121
45	114	13	127
46	114	13	127
47	103	10	113
48	106	12	118
49	105	10	115
50	125	13	138
51	119	10	129
52	130	15	145
53	101	11	112
54	128	15	143
55	116	9	125
56	121	14	135
57	120	12	132
58	133	10	143
59	123	9	132
60	117	10	127
61	110	12	122
62	114	9	123
63	92	16	108
64	70	8	78
65	66	14	80
66	24	0	24
67	12	0	12
All	4520	480	5000



In [518…
```
sns.displot(data=df, x="Age", hue="Personal_Loan",
            height=8.07, aspect=11.7/8.27
           );
```



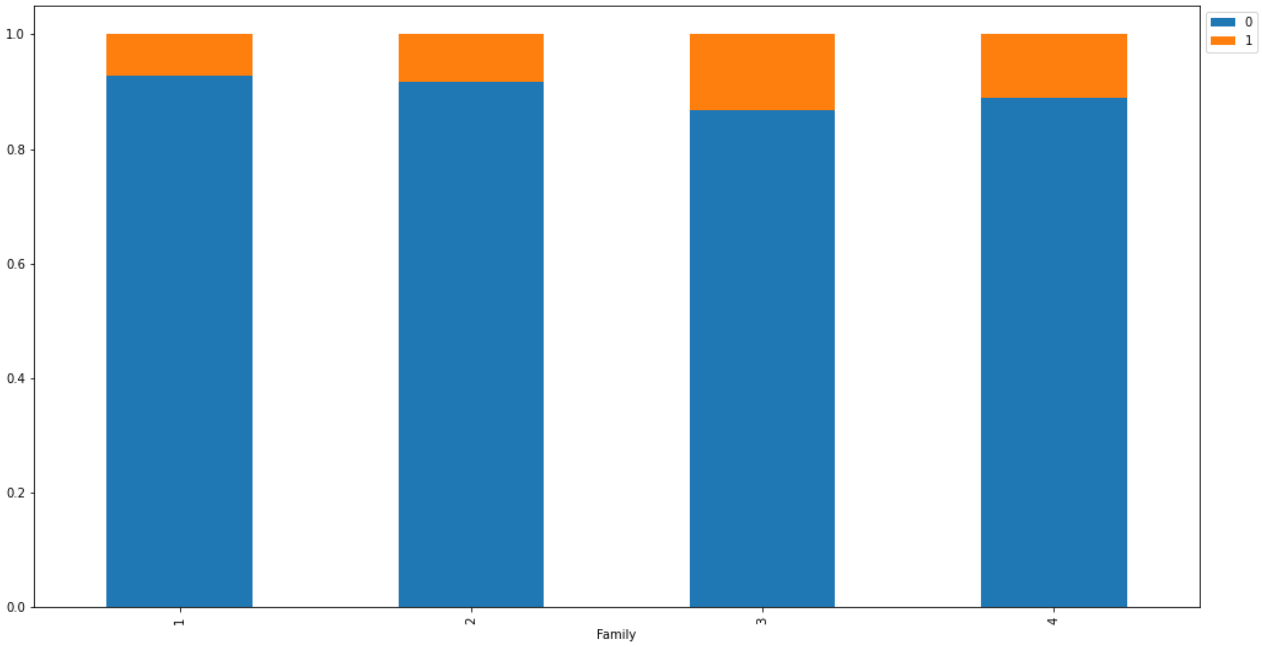
- Customer between ages 26 and 65 converted to Asset Customers
- Distribution of age is even.

Family vs Personal_Loan

In [518…
```
tab1 = pd.crosstab(df.Family , df.Personal_Loan , margins=True)
print(tab1)
```

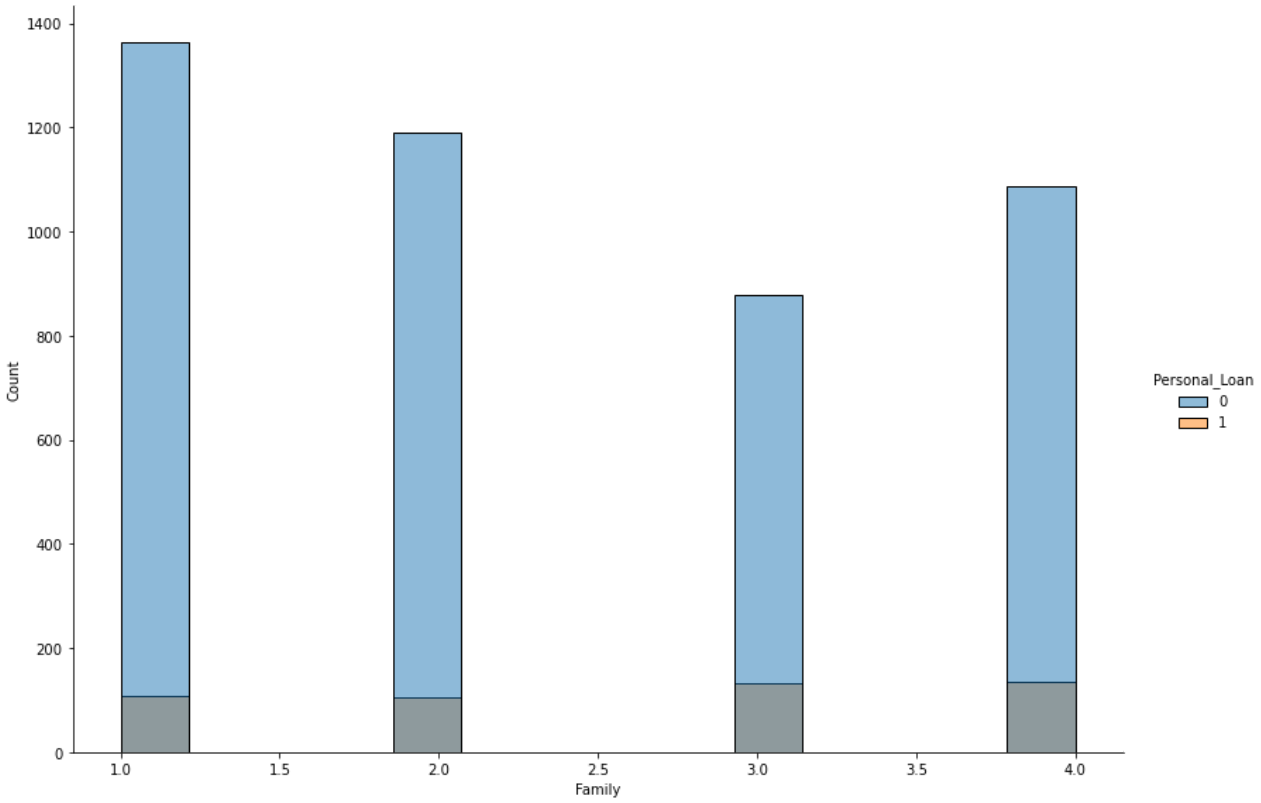
```
print('-'*120)
tab = pd.crosstab(df.Family , df.Personal_Loan , normalize='index')
tab.plot(kind='bar',stacked=True,figsize=(17,9))
plt.legend(loc="upper left", bbox_to_anchor=(1,1));
```

```
Personal_Loan     0    1   All
Family
1              1365  107  1472
2              1190  106  1296
3               877  133  1010
4              1088  134  1222
All            4520  480  5000
------------------------------------------------------------------------------------------------------------
--------------------------------
```



In [518…

```
sns.displot(data=df, x="Family", hue="Personal_Loan",
            height=8.07, aspect=11.7/8.27
           );
```



- Family with 3-4 members are more inclined to convert to Asset Customers 12%(avg) versus 8% of Family with 1-2 members.

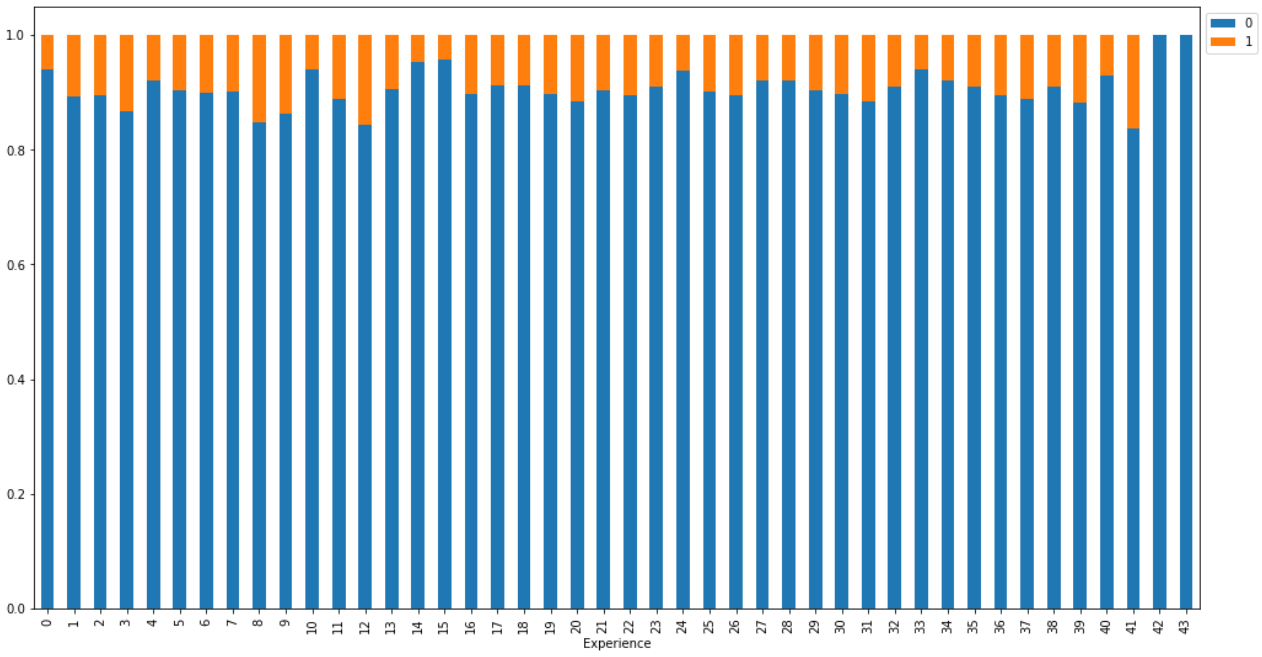
Experience vs Personal_Loan

In [518…

```
tab1 = pd.crosstab(df.Experience , df.Personal_Loan , margins=True)
print(tab1)
print('-'*120)
tab = pd.crosstab(df.Experience , df.Personal_Loan , normalize='index')
tab.plot(kind='bar',stacked=True,figsize=(17,9))
plt.legend(loc="upper left", bbox_to_anchor=(1,1));
```

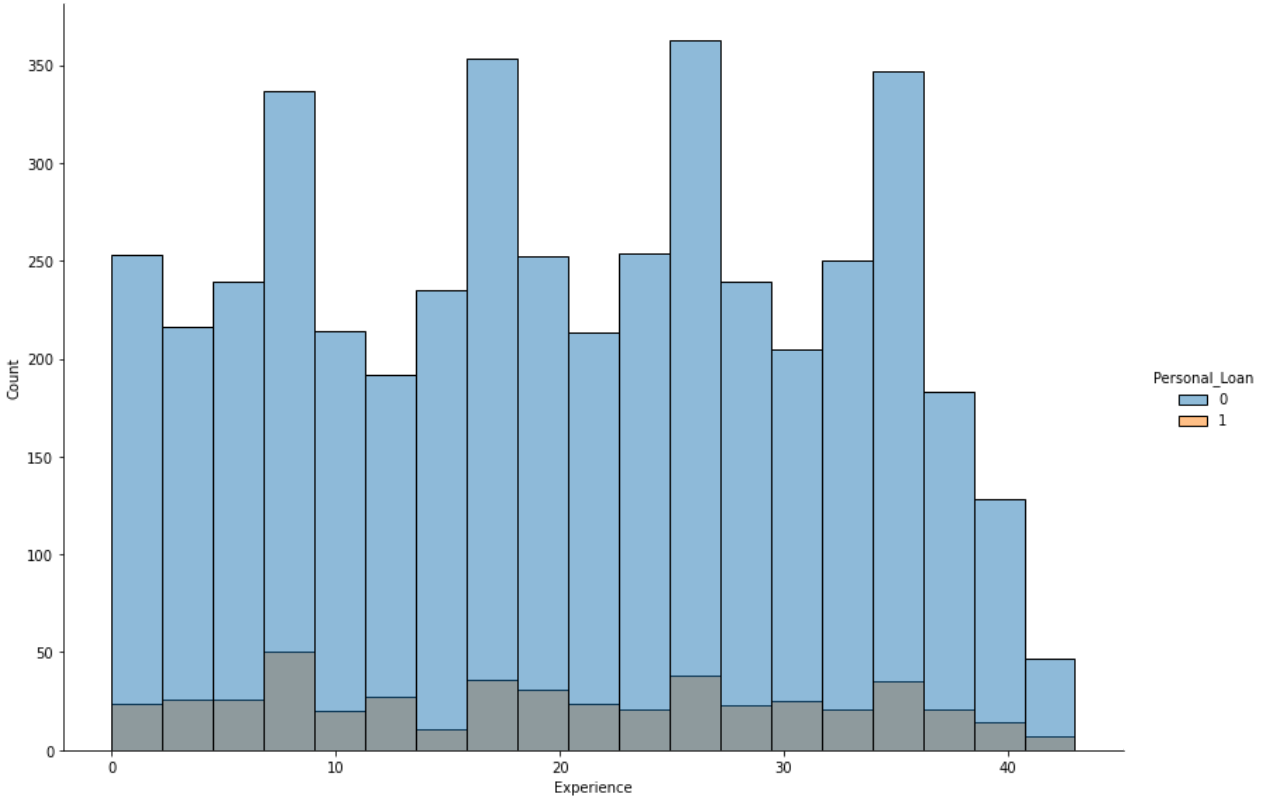
```
Personal_Loan     0    1   All
Experience
0               111    7   118
1                66    8    74
2                76    9    85
3               112   17   129
4               104    9   113
5               132   14   146
6               107   12   119
7               109   12   121
8               101   18   119
9               127   20   147
10              111    7   118
11              103   13   116
12               86   16   102
13              106   11   117
14              121    6   127
15              114    5   119
16              114   13   127
17              114   11   125
18              125   12   137
19              121   14   135
```

```
20                131   17   148
21                102   11   113
22                111   13   124
23                131   13   144
24                123    8   131
25                128   14   142
26                120   14   134
27                115   10   125
28                127   11   138
29                112   12   124
30                113   13   126
31                 92   12   104
32                140   14   154
33                110    7   117
34                115   10   125
35                130   13   143
36                102   12   114
37                103   13   116
38                 80    8    88
39                 75   10    85
40                 53    4    57
41                 36    7    43
42                  8    0     8
43                  3    0     3
All              4520  480  5000
```



In [518…

```
sns.displot(data=df, x="Experience", hue="Personal_Loan",
            height=8.07, aspect=11.7/8.27
           );
```



- Customer with 0 to 41 years `Experience` converted to Asset Customers.
- Distribution of `Experience` is even and proportional to the total amount per level of experience.

Income vs Personal_Loan

In [518…

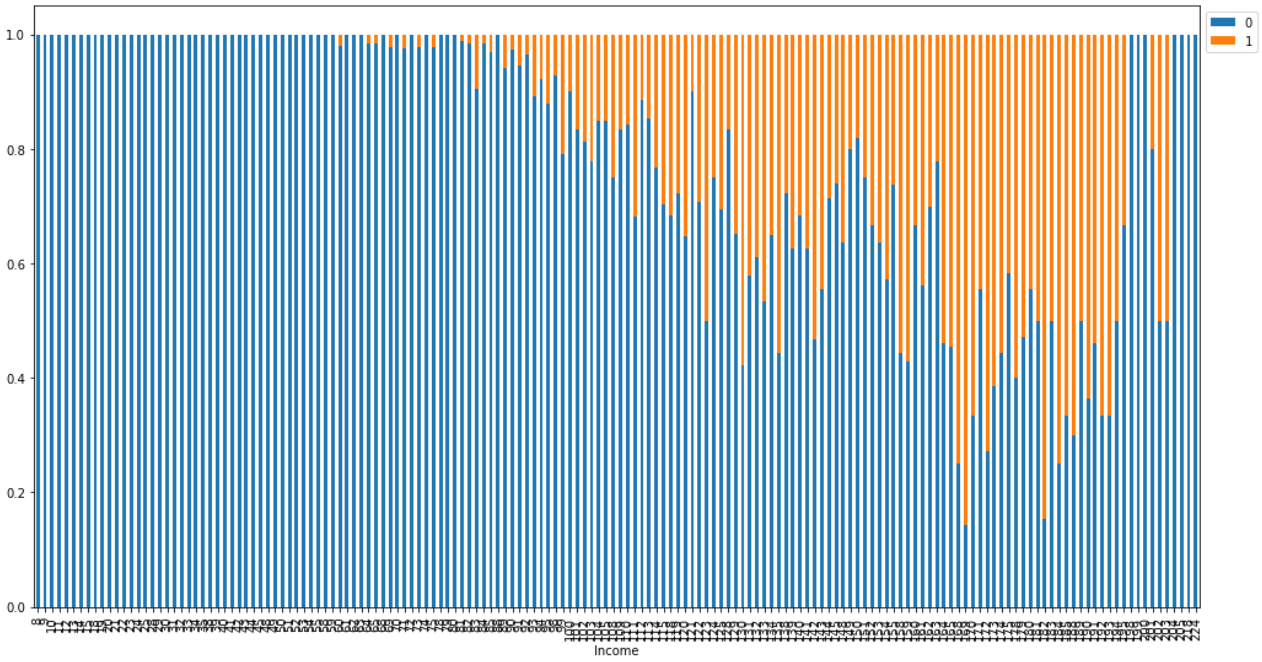
```
tab1 = pd.crosstab(df.Income , df.Personal_Loan , margins=True)
print(tab1)
print('-'*120)
tab = pd.crosstab(df.Income , df.Personal_Loan , normalize='index')
tab.plot(kind='bar',stacked=True,figsize=(17,9))
plt.legend(loc="upper left", bbox_to_anchor=(1,1));
```

```
Personal_Loan     0    1   All
Income
8                23    0    23
9                26    0    26
10               23    0    23
11               27    0    27
12               30    0    30
13               32    0    32
14               31    0    31
15               33    0    33
18               53    0    53
19               52    0    52
20               47    0    47
21               65    0    65
22               65    0    65
23               54    0    54
24               47    0    47
25               64    0    64
28               63    0    63
29               67    0    67
30               63    0    63
```

31	55	0	55
32	58	0	58
33	51	0	51
34	53	0	53
35	65	0	65
38	84	0	84
39	81	0	81
40	78	0	78
41	82	0	82
42	77	0	77
43	70	0	70
44	85	0	85
45	69	0	69
48	44	0	44
49	52	0	52
50	45	0	45
51	41	0	41
52	47	0	47
53	57	0	57
54	52	0	52
55	61	0	61
58	55	0	55
59	53	0	53
60	51	1	52
61	57	0	57
62	55	0	55
63	46	0	46
64	59	1	60
65	59	1	60
68	35	0	35
69	45	1	46
70	47	0	47
71	42	1	43
72	41	0	41
73	43	1	44
74	45	0	45
75	46	1	47
78	61	0	61
79	53	0	53
80	56	0	56
81	82	1	83
82	60	1	61
83	67	7	74
84	62	1	63
85	63	2	65
88	26	0	26
89	32	2	34
90	37	1	38
91	35	2	37
92	28	1	29
93	33	4	37
94	24	2	26
95	22	3	25
98	26	2	28
99	19	5	24
100	9	1	10
101	20	4	24
102	13	3	16
103	14	4	18
104	17	3	20
105	17	3	20
108	12	4	16
109	15	3	18
110	16	3	19
111	15	7	22

112	23	3	26
113	29	5	34
114	23	7	30
115	19	8	27
118	13	6	19
119	13	5	18
120	11	6	17
121	18	2	20
122	17	7	24
123	9	9	18
124	9	3	12
125	16	7	23
128	20	4	24
129	15	8	23
130	8	11	19
131	11	8	19
132	11	7	18
133	8	7	15
134	13	7	20
135	8	10	18
138	13	5	18
139	10	6	16
140	13	6	19
141	15	9	24
142	7	8	15
143	5	4	9
144	5	2	7
145	17	6	23
148	7	4	11
149	16	4	20
150	9	2	11
151	3	1	4
152	10	5	15
153	7	4	11
154	12	9	21
155	14	5	19
158	8	10	18
159	3	4	7
160	8	4	12
161	9	7	16
162	7	3	10
163	7	2	9
164	6	7	13
165	5	6	11
168	2	6	8
169	1	6	7
170	4	8	12
171	5	4	9
172	3	8	11
173	5	8	13
174	4	5	9
175	7	5	12
178	4	6	10
179	8	9	17
180	10	8	18
181	4	4	8
182	2	11	13
183	6	6	12
184	3	9	12
185	3	6	9
188	3	7	10
189	1	1	2
190	4	7	11
191	6	7	13
192	2	4	6

```
193               2    4     6
194               4    4     8
195              10    5    15
198               3    0     3
199               3    0     3
200               3    0     3
201               4    1     5
202               1    1     2
203               1    1     2
204               3    0     3
205               2    0     2
218               1    0     1
224               1    0     1
All            4520  480  5000
```



- It looks like the tendency for customer with higher `Income` will convert to Asset Customer.

In [ ]:

CCAvg vs Personal_Loan

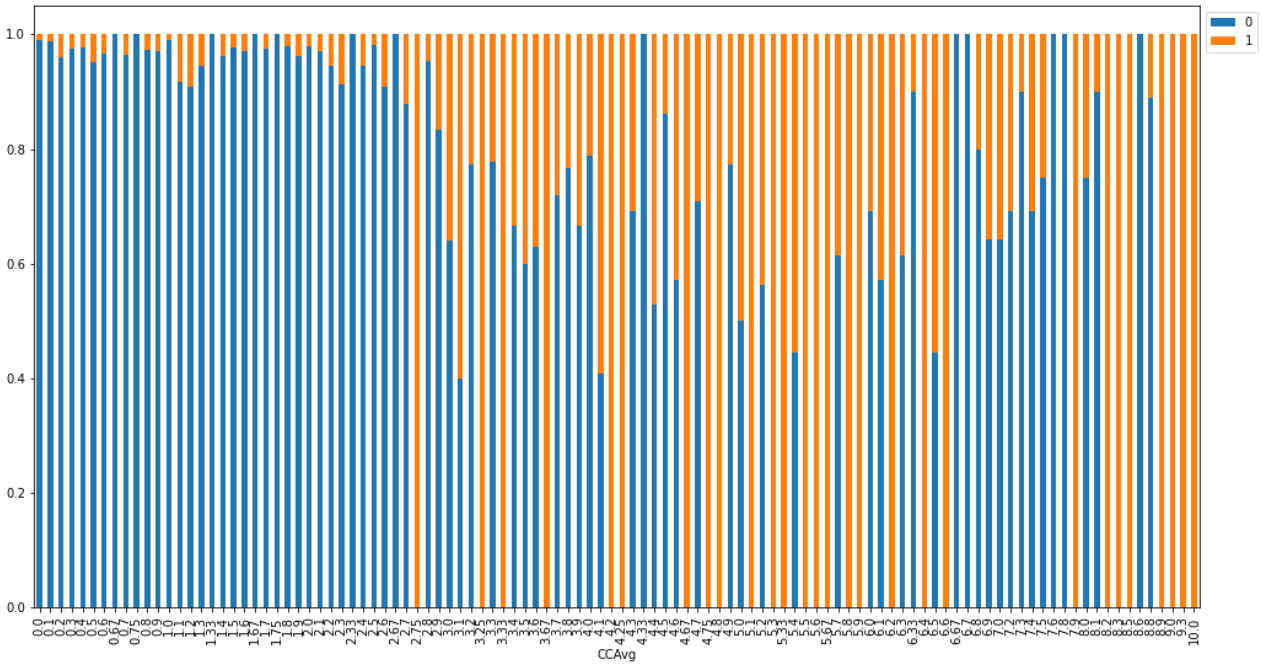
In [519…

```
tab1 = pd.crosstab(df.CCAvg , df.Personal_Loan , margins=True)
print(tab1)
print('-'*120)
tab = pd.crosstab(df.CCAvg , df.Personal_Loan , normalize='index')
tab.plot(kind='bar',stacked=True,figsize=(17,9))
plt.legend(loc="upper left", bbox_to_anchor=(1,1));
```

```
Personal_Loan     0    1   All
CCAvg
0.0             105    1   106
0.1             181    2   183
0.2             196    8   204
0.3             235    6   241
0.4             175    4   179
0.5             155    8   163
0.6             114    4   118
0.67             18    0    18
```

0.7	163	6	169
0.75	9	0	9
0.8	182	5	187
0.9	103	3	106
1.0	229	2	231
1.1	77	7	84
1.2	60	6	66
1.3	121	7	128
1.33	9	0	9
1.4	131	5	136
1.5	174	4	178
1.6	98	3	101
1.67	18	0	18
1.7	154	4	158
1.75	9	0	9
1.8	149	3	152
1.9	102	4	106
2.0	184	4	188
2.1	97	3	100
2.2	123	7	130
2.3	53	5	58
2.33	18	0	18
2.4	87	5	92
2.5	105	2	107
2.6	79	8	87
2.67	36	0	36
2.7	51	7	58
2.75	0	1	1
2.8	105	5	110
2.9	45	9	54
3.0	34	19	53
3.1	8	12	20
3.2	17	5	22
3.25	0	1	1
3.3	35	10	45
3.33	0	1	1
3.4	26	13	39
3.5	9	6	15
3.6	17	10	27
3.67	0	1	1
3.7	18	7	25
3.8	33	10	43
3.9	18	9	27
4.0	26	7	33
4.1	9	13	22
4.2	0	11	11
4.25	0	2	2
4.3	18	8	26
4.33	9	0	9
4.4	9	8	17
4.5	25	4	29
4.6	8	6	14
4.67	0	1	1
4.7	17	7	24
4.75	0	2	2
4.8	0	7	7
4.9	17	5	22
5.0	9	9	18
5.1	0	6	6
5.2	9	7	16
5.3	0	4	4
5.33	0	1	1
5.4	8	10	18
5.5	0	4	4
5.6	0	7	7

```
5.67              0    2     2
5.7               8    5    13
5.8               0    3     3
5.9               0    5     5
6.0              18    8    26
6.1               8    6    14
6.2               0    2     2
6.3               8    5    13
6.33              9    1    10
6.4               0    3     3
6.5               8   10    18
6.6               0    4     4
6.67              9    0     9
6.7               9    0     9
6.8               8    2    10
6.9               9    5    14
7.0               9    5    14
7.2               9    4    13
7.3               9    1    10
7.4               9    4    13
7.5               9    3    12
7.6               9    0     9
7.8               9    0     9
7.9               0    4     4
8.0               9    3    12
8.1               9    1    10
8.2               0    1     1
8.3               0    2     2
8.5               0    2     2
8.6               8    0     8
8.8               8    1     9
8.9               0    1     1
9.0               0    2     2
9.3               0    1     1
10.0              0    3     3
All            4520  480  5000
```



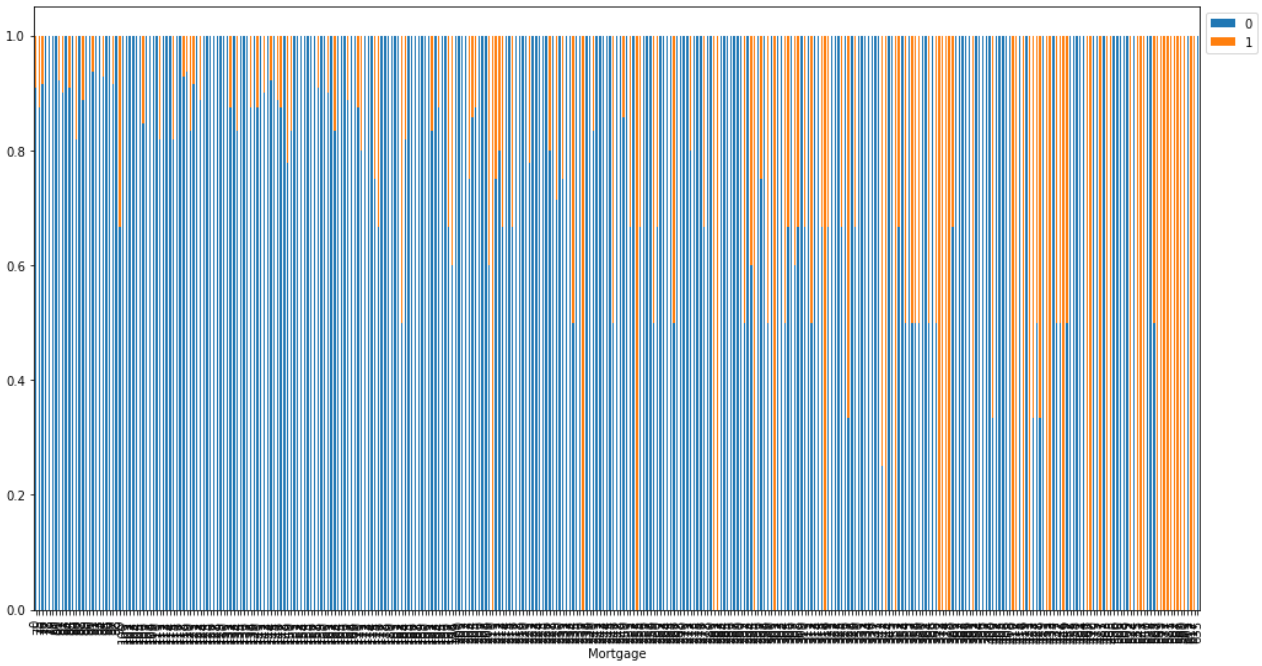
- This one is not that clear, but it looks like customer with higher `CCAvg` will convert to Asset Customer. with some exceptions.

Mortgage vs Personal_Loan

In [519…

```
tab1 = pd.crosstab(df.Mortgage , df.Personal_Loan , margins=True)
print(tab1)
print('-'*120)
tab = pd.crosstab(df.Mortgage , df.Personal_Loan , normalize='index')
tab.plot(kind='bar',stacked=True,figsize=(17,9))
plt.legend(loc="upper left", bbox_to_anchor=(1,1));
```

```
Personal_Loan     0    1   All
Mortgage
0              3150  312  3462
75                7    1     8
76               11    1    12
77                4    0     4
78               15    0    15
...             ...  ...   ...
601               1    0     1
612               0    1     1
617               0    1     1
635               1    0     1
All            4520  480  5000

[348 rows x 3 columns]
------------------------------------------------------------------------------------------------------------
--------------------------------
```

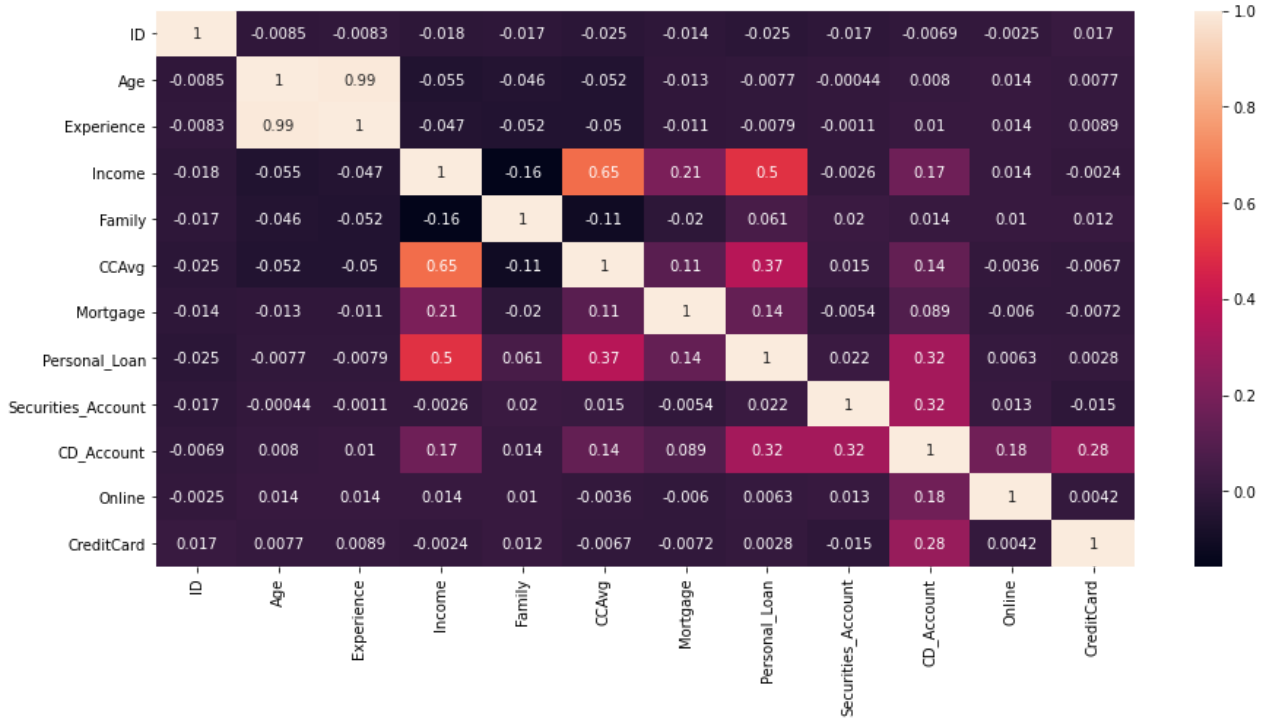


- Same as in `CCAvg` , this one is not that clear, but it looks like customer with higher `Mortgage` will convert to Asset Customer. with some exceptions.

Correlation between numeric Variables

In [519…

```
plt.figure(figsize=(15,7))
sns.heatmap(df.corr(),annot=True)
plt.show()
```



We will investigate further those valiables that have high correlation with `Personal_Loan` .

Evaluate the variables that have .15 or higher correlation point vs Personal_Loan:

- Experience vs Age vs Personal_Loan
- Income vs Mortgage vs Personal_Loan
- Income vs CCAvg vs Personal_Loan
- Income vs CD_Account vs Personal_Loan
- CD_Account vs Securities_Account vs Personal_Loan
- CD_Account vs CreditCard vs Personal_Loan

PairPlot

In [519…

```
sns.pairplot(data=df,hue="Personal_Loan",)
plt.show()
```

In [ ]:

In [ ]:

Multivariate Analysis

- We will analyze further the variables that have a high correlation with `Personal_Loan`

In [519…

```
def box_cat_vs_num( dataframe , categorical , numerical, hue=None , palette=None , figs
    plt.figure(figsize=figsize)
    if palette:
        sns.boxplot( dataframe[categorical] , dataframe[numerical] , palette=palette)
    else:
        sns.boxplot( dataframe[categorical] , dataframe[numerical] )
    plt.xlabel(categorical)
    plt.ylabel(numerical)

    if hue:
        plt.figure(figsize=figsize)
        if palette:
```

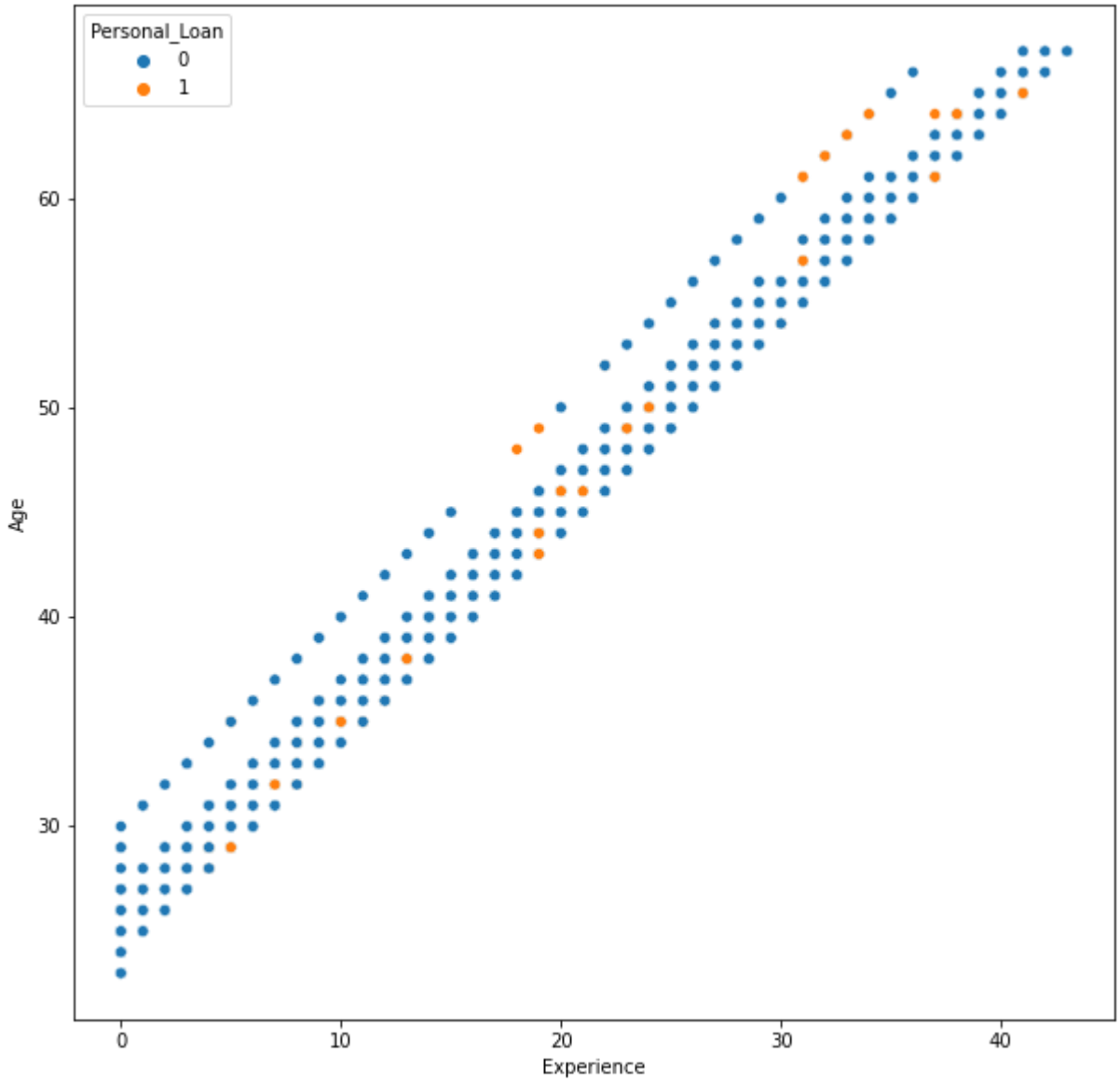
```
            sns.boxplot(data=dataframe , x=categorical , y=numerical , hue=hue , palett
        else:
            sns.boxplot(data=dataframe , x=categorical , y=numerical , hue=hue)
        plt.show()

        plt.figure(figsize=figsize)

        #yint = range(min( dataframe[numerical] ), math.ceil(max(  dataframe[numerical]

       #plt.yticks(yint)

        if palette:
            sns.barplot(data=dataframe , x=categorical , y=numerical , hue=hue , units=
        else:
            sns.barplot(data=dataframe , x=categorical , y=numerical , hue=hue, units=1

        plt.show()
```

In [ ]:

Experience vs Age vs Personal_Loan

In [519…

```
plt.figure(figsize = (10,10))
sns.scatterplot(x = df['Experience'], y = df['Age'] , hue = df['Personal_Loan'])
plt.show()
```



- Correlation is not necessarily high with `Personal_Loan` , but it is high between `Experience` and `Age` . This is logical since Age and Experience usually go together. We will need to chack for any Multicollinearity further on.

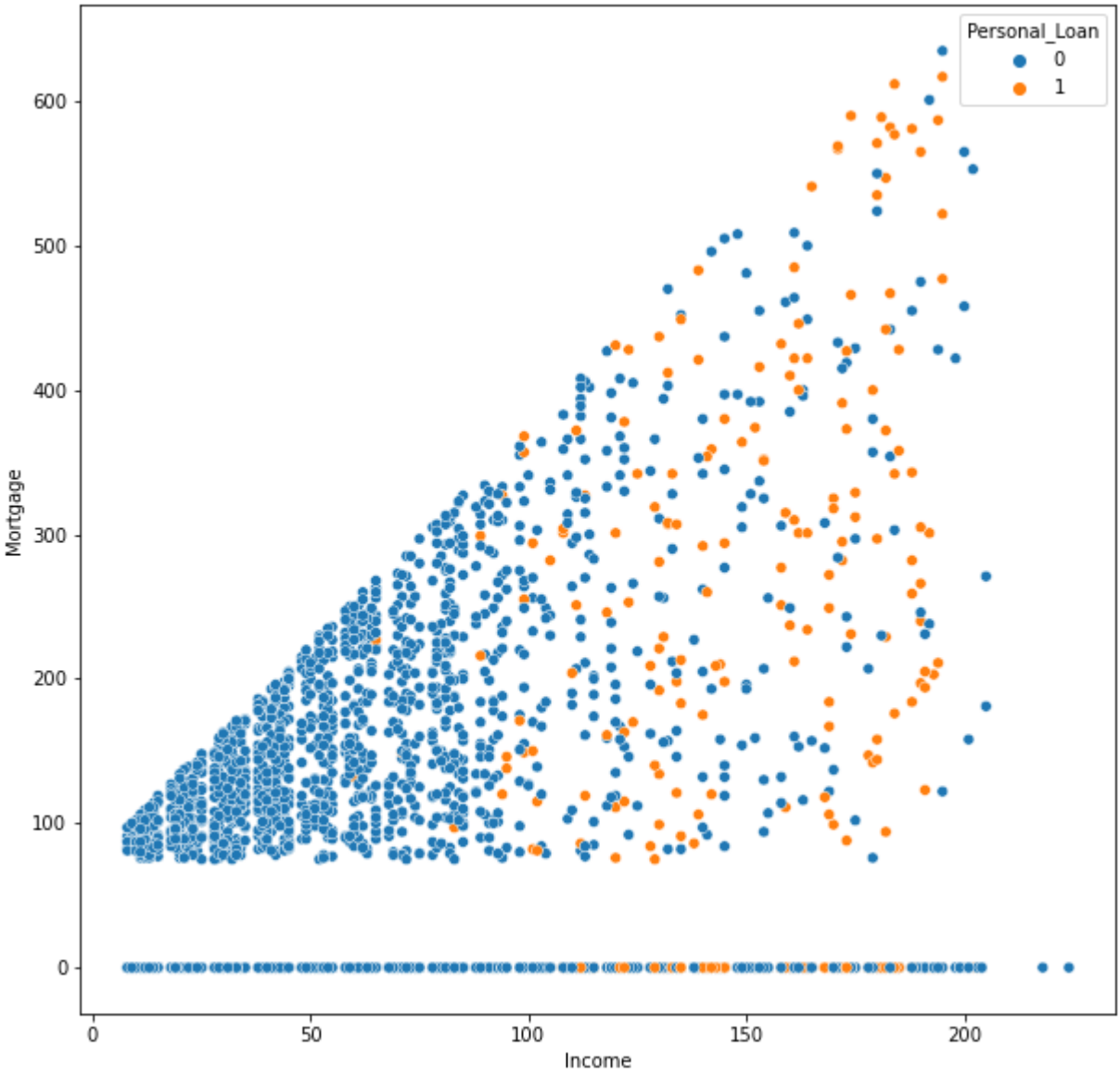
In [ ]:

In [ ]:

Income vs Mortgage vs Personal_Loan

In [519…

```
plt.figure(figsize = (10,10))
sns.scatterplot(x = df['Income'], y = df['Mortgage'] , hue = df['Personal_Loan'])
plt.show()
```

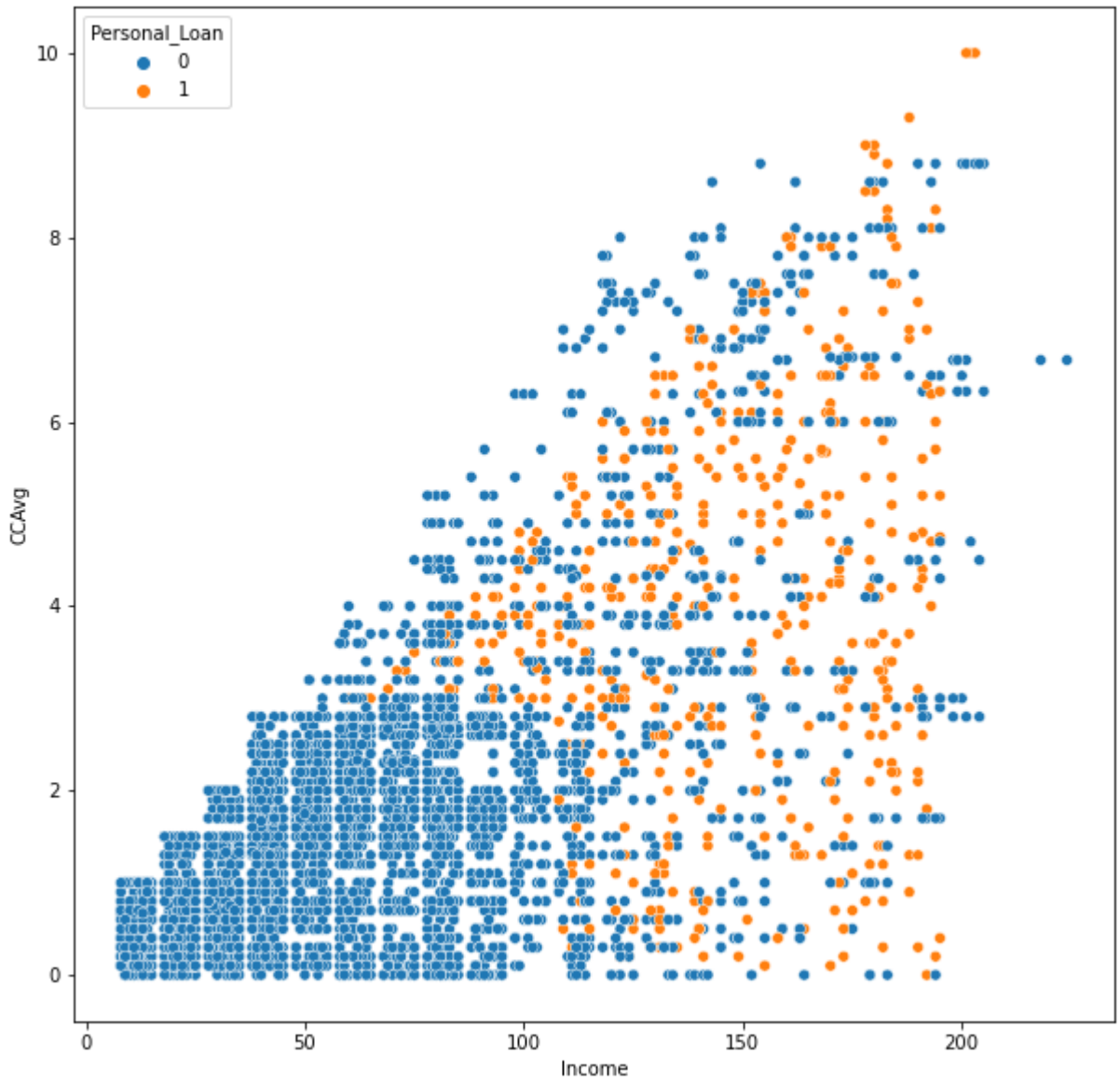


- This shows that customer with higher Incomes and higher Mortgage have a correlation with customer that took `Personal_Loan'. As opposed to those customers that have lower Income and Mortgage, which did not take the Personal_Loan.
- Income and Mortgage may need to be checked for Multicollinearity.

Income vs CCAvg vs Personal_Loan

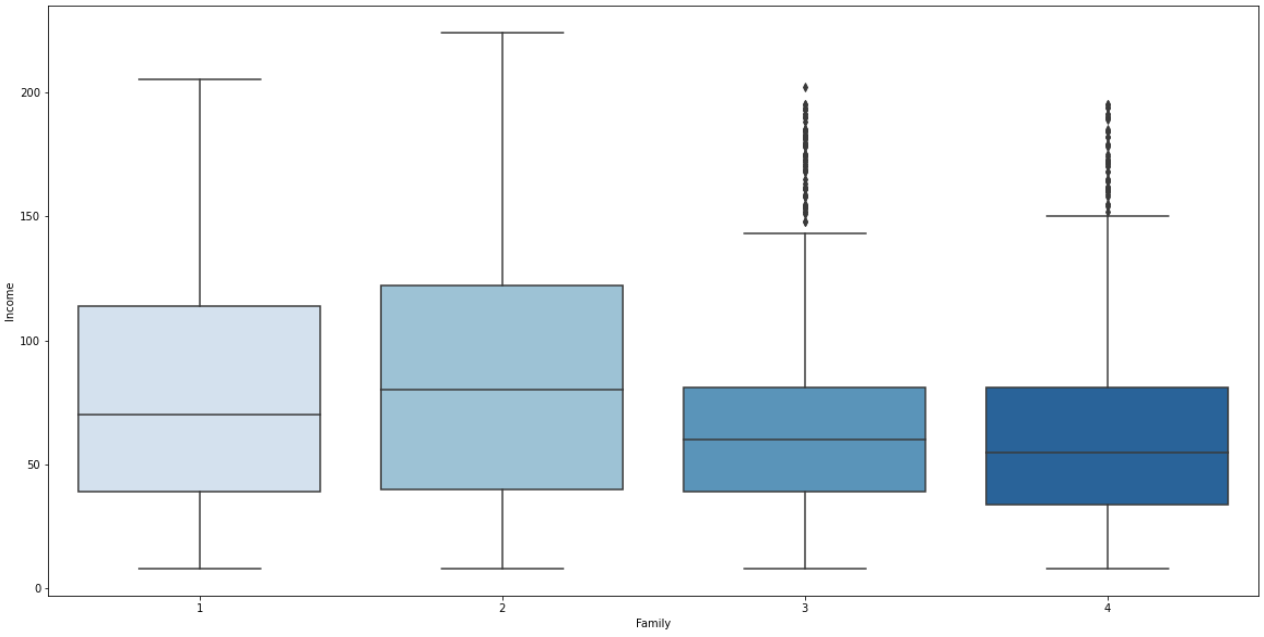
In [519…

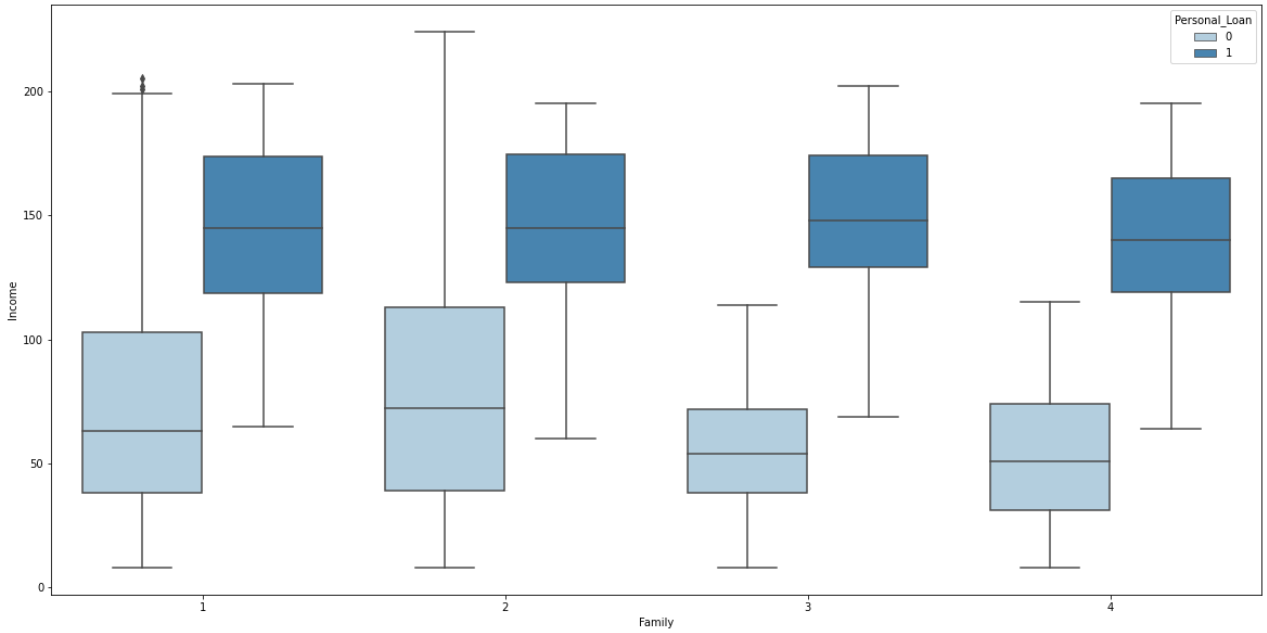
```
plt.figure(figsize = (10,10))
sns.scatterplot(x = df['Income'], y = df['CCAvg'] , hue = df['Personal_Loan'])
plt.show()
```

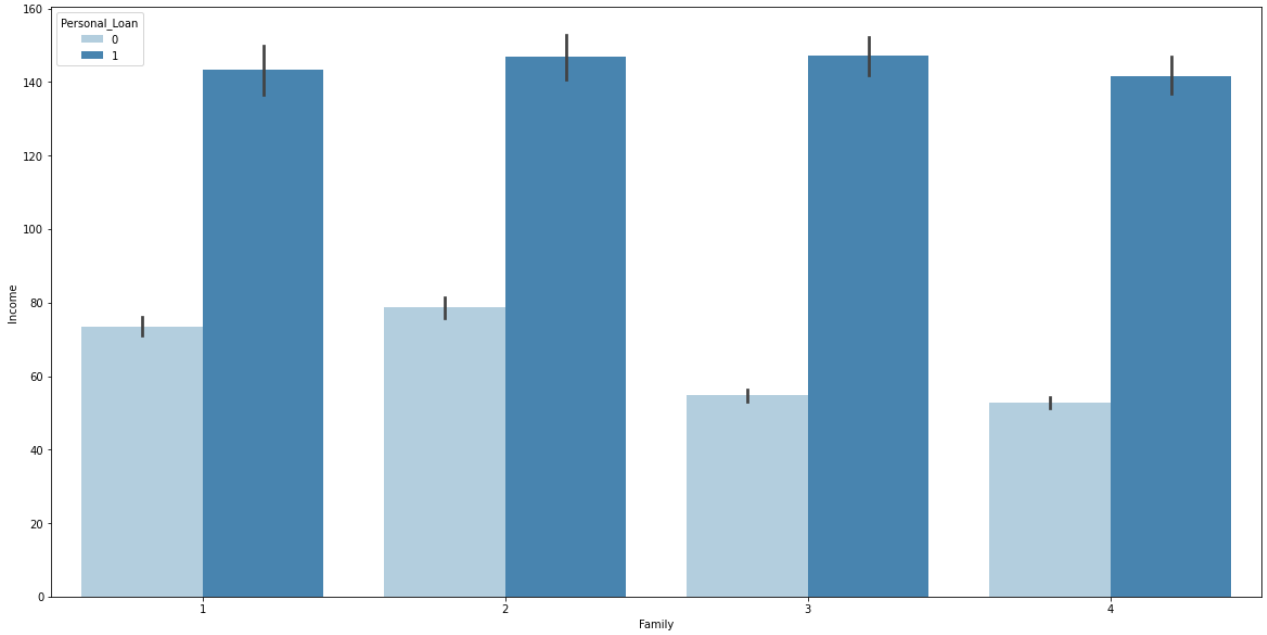


In [519…

```
box_cat_vs_num( df , 'Family' , 'Income' , 'Personal_Loan' , 'Blues')
```







- This shows that customer with higher Incomes and higher CC_Avg have a correlation with customer that took `Personal_Loan'. As opposed to those customers that have lower Income and CC_Avg, which did not take the Personal_Loan.
- Income and CCAvg may need to be checked for Multicollinearity.

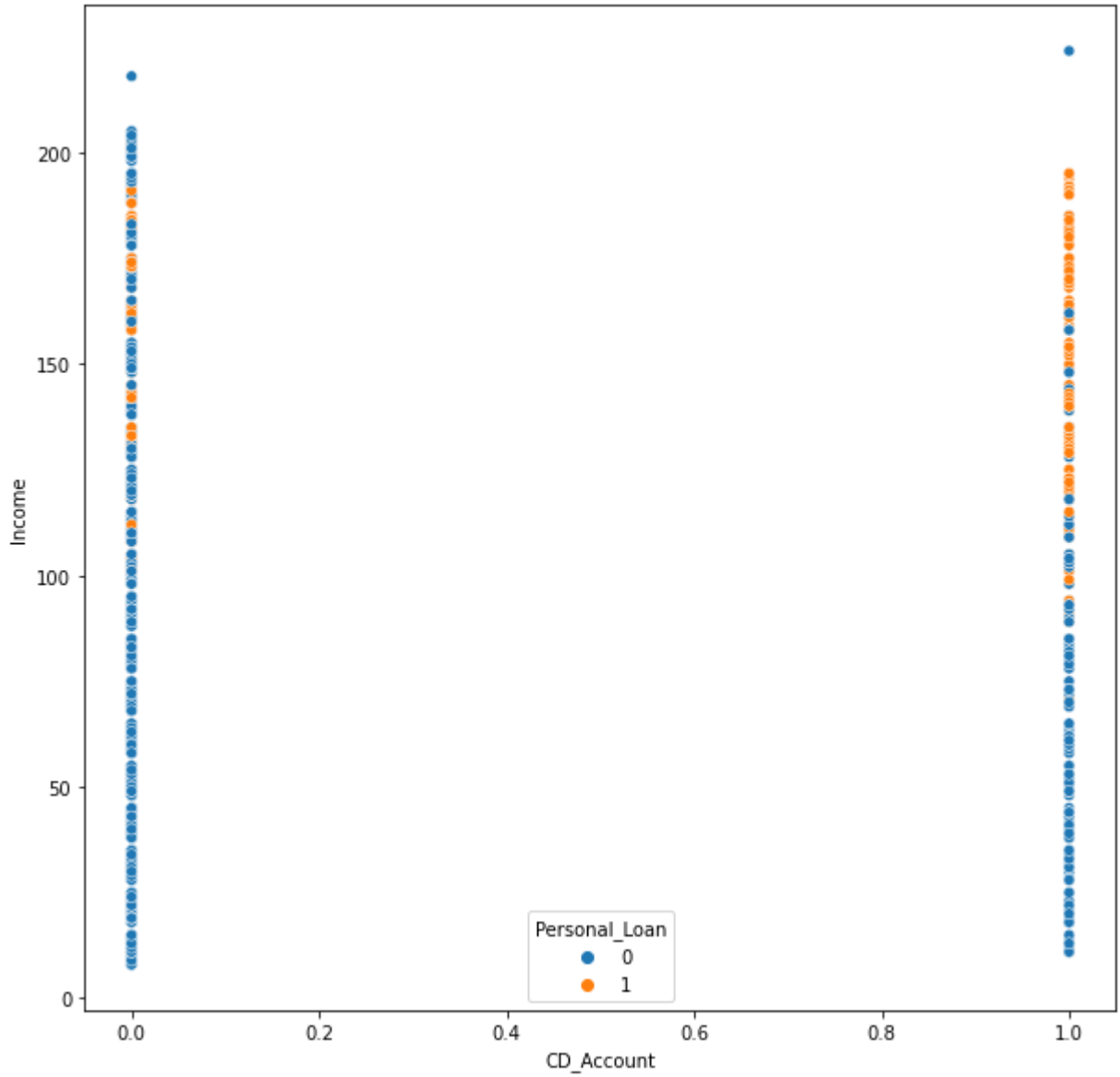
In [ ]:

In [ ]:

Income vs CD_Account vs Personal_Loan

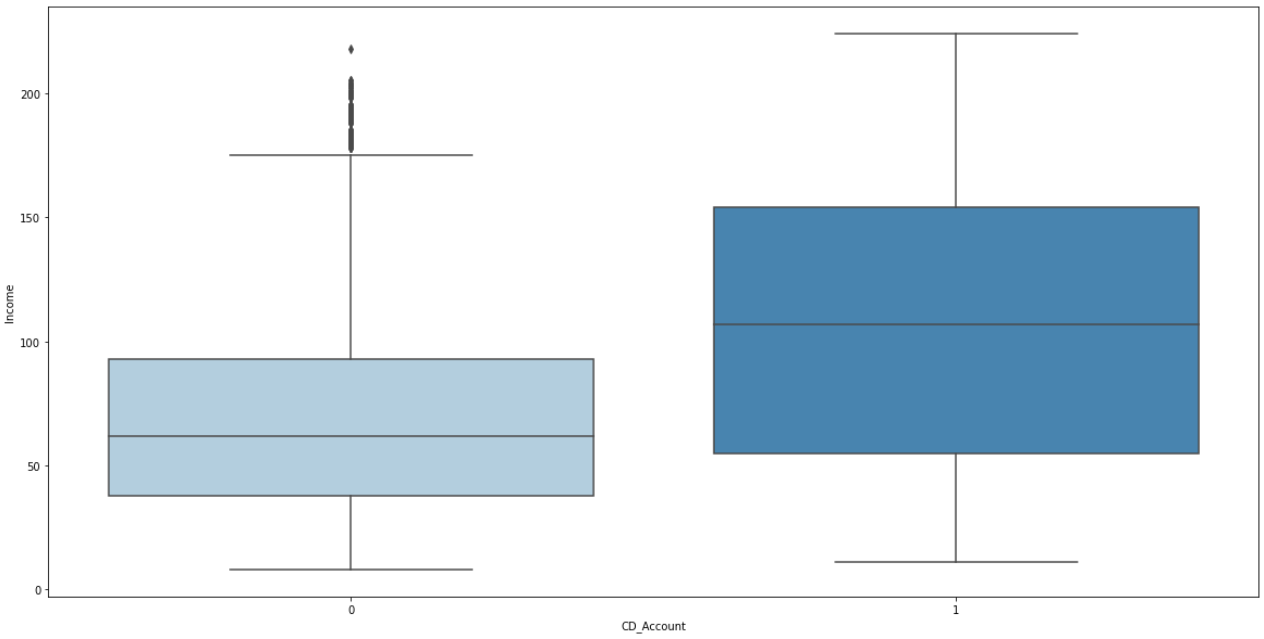
In [519…

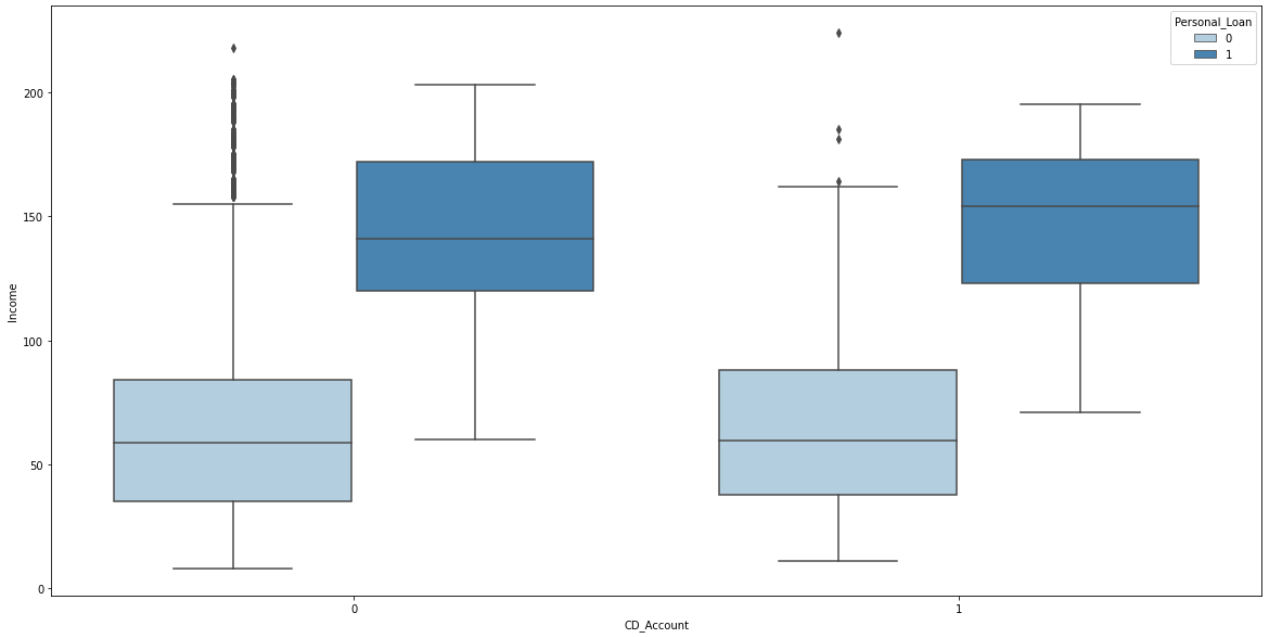
```
plt.figure(figsize = (10,10))
sns.scatterplot(x = df['CD_Account'], y = df['Income'] , hue = df['Personal_Loan'])
plt.show()
```

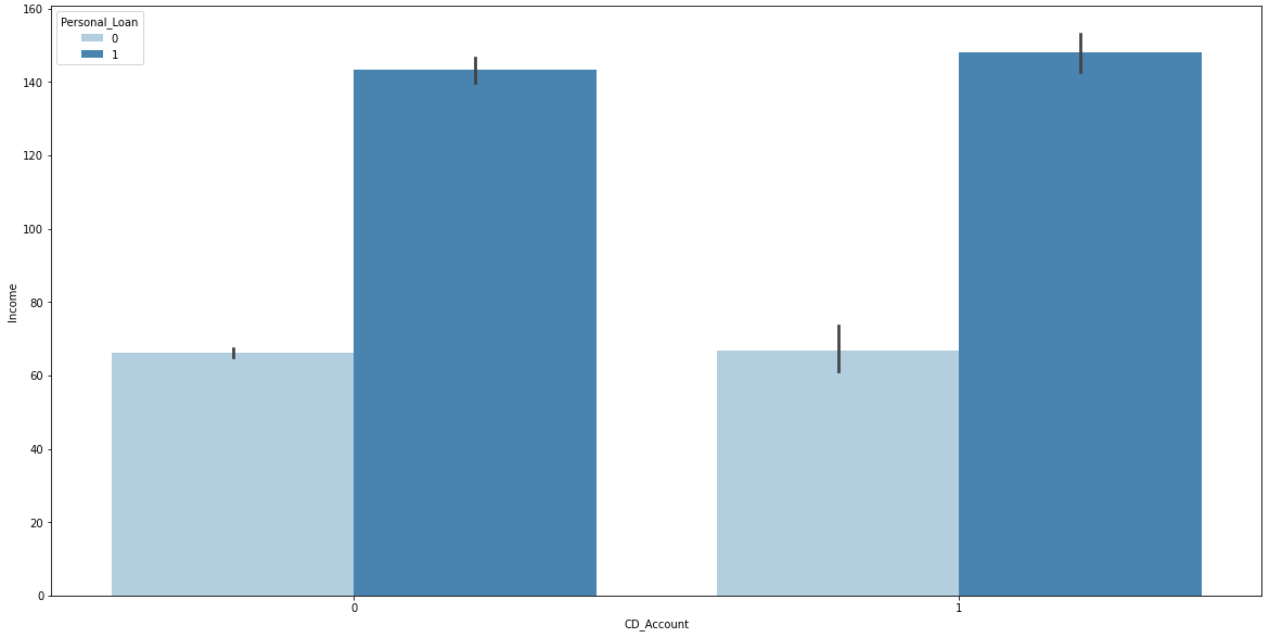


In [520…

```
box_cat_vs_num( df , 'CD_Account' , 'Income' , 'Personal_Loan' , 'Blues')
```







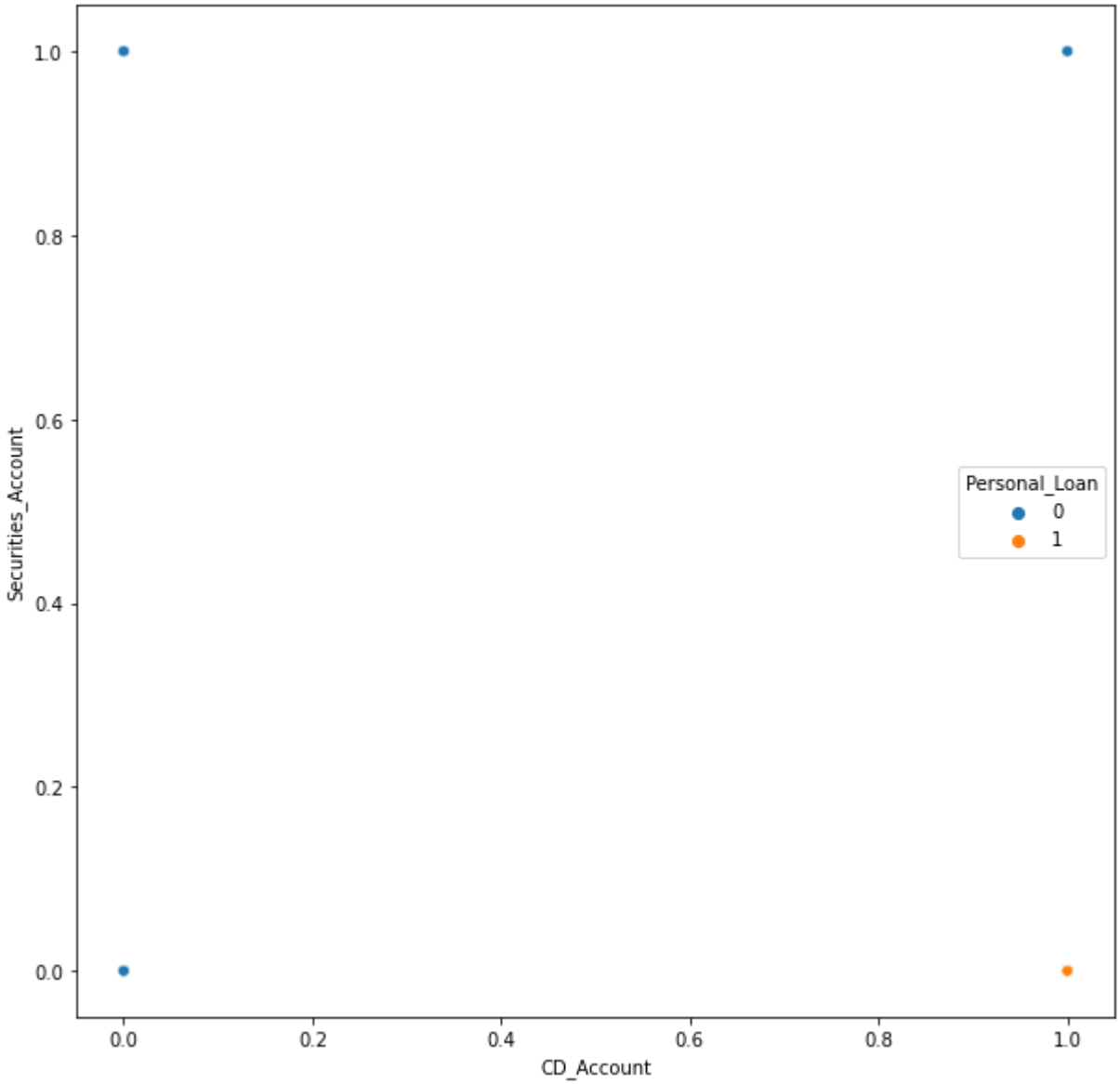
- There is high correlation for customers `Income` and have a CD_Account for customer that opened a Personal_Loan.

In [ ]:

CD_Account vs Securities_Account vs Personal_Loan

In [520…

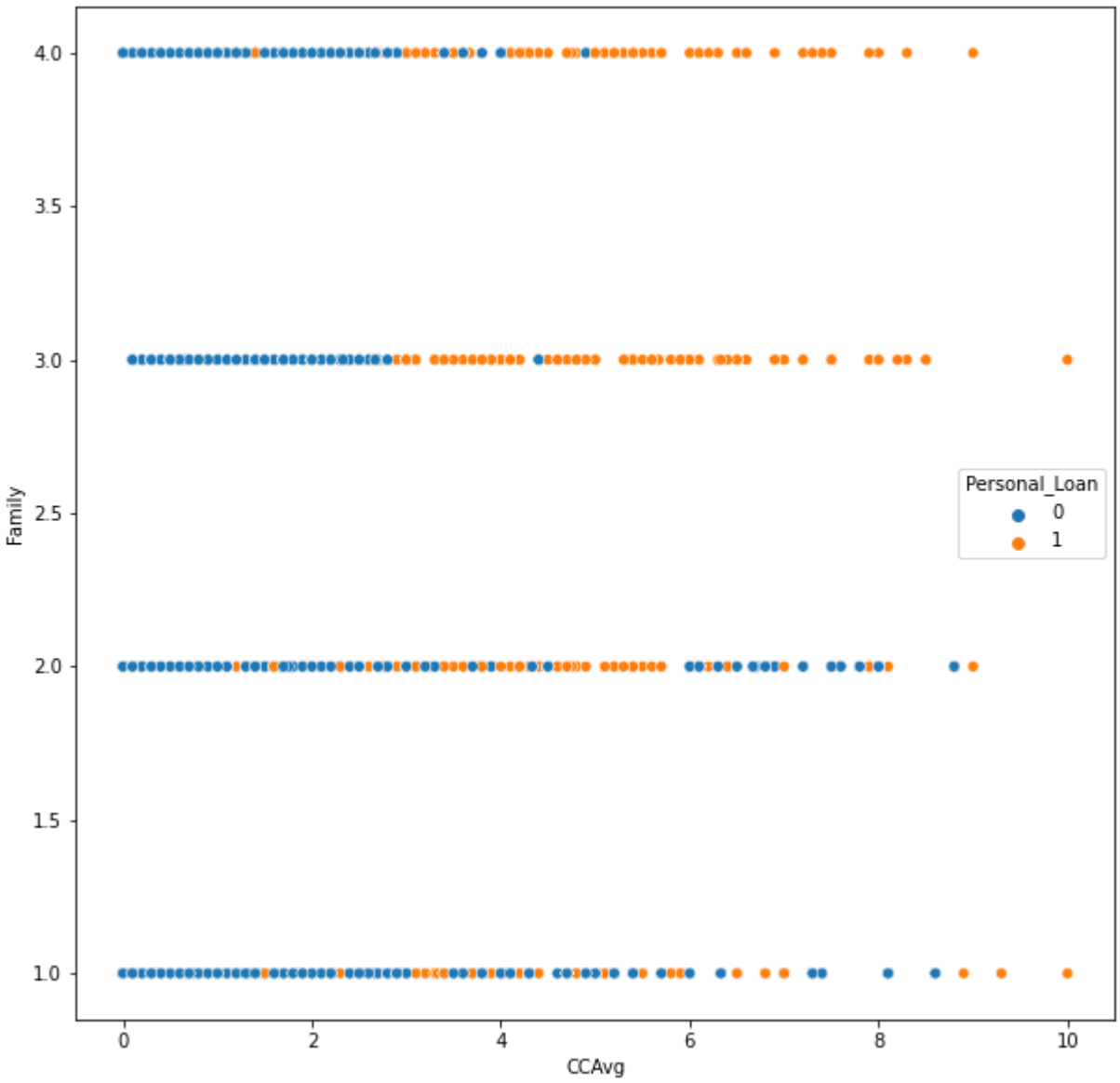
```
plt.figure(figsize = (10,10))
sns.scatterplot(x = df['CD_Account'], y = df['Securities_Account'] , hue = df['Personal
plt.show()
```



Family vs CCAvg vs Personal_Loan

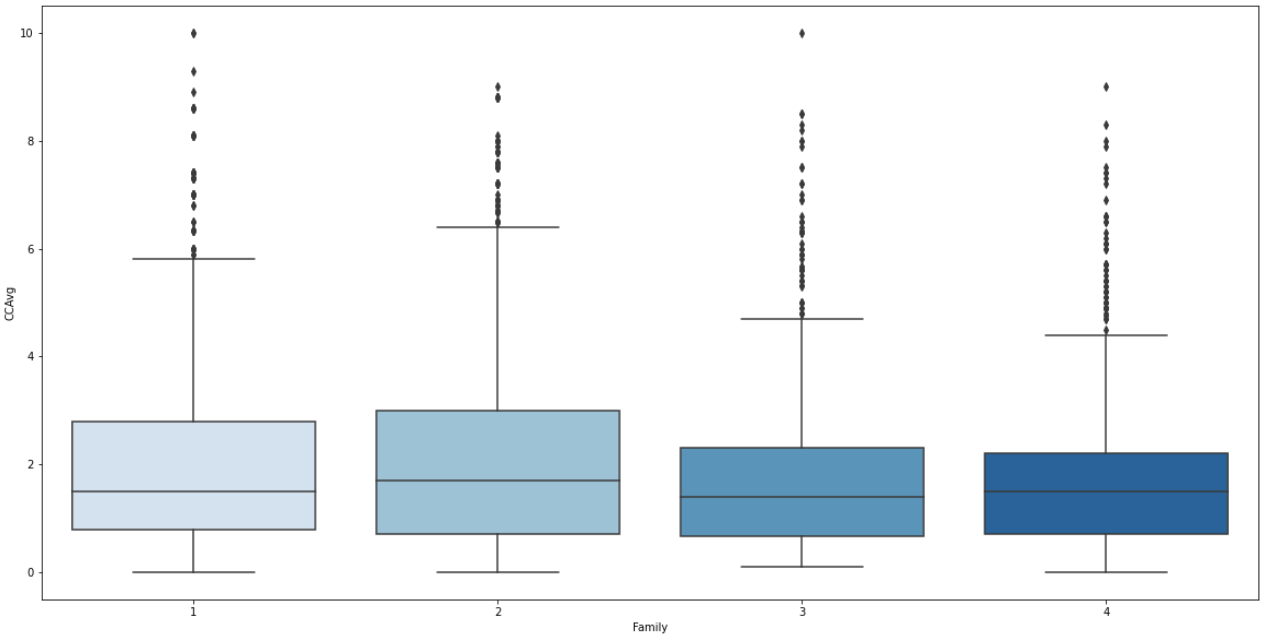
In [520…

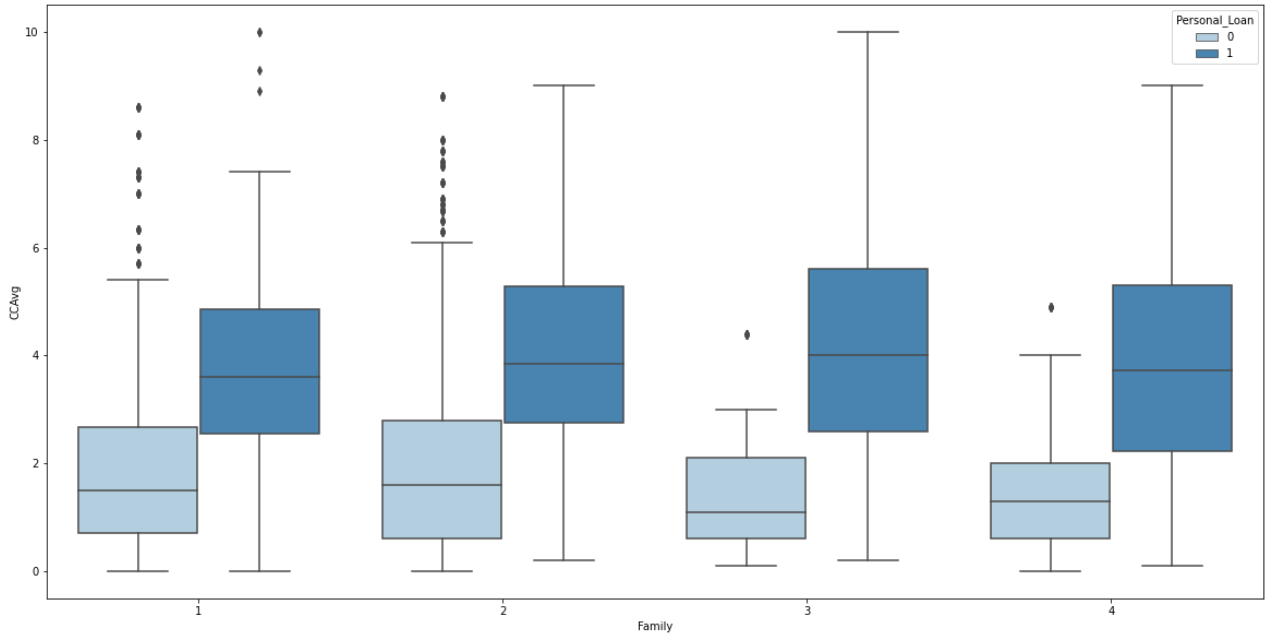
```
plt.figure(figsize = (10,10))
sns.scatterplot(x = df['CCAvg'], y = df['Family'] , hue = df['Personal_Loan'])
plt.show()
```

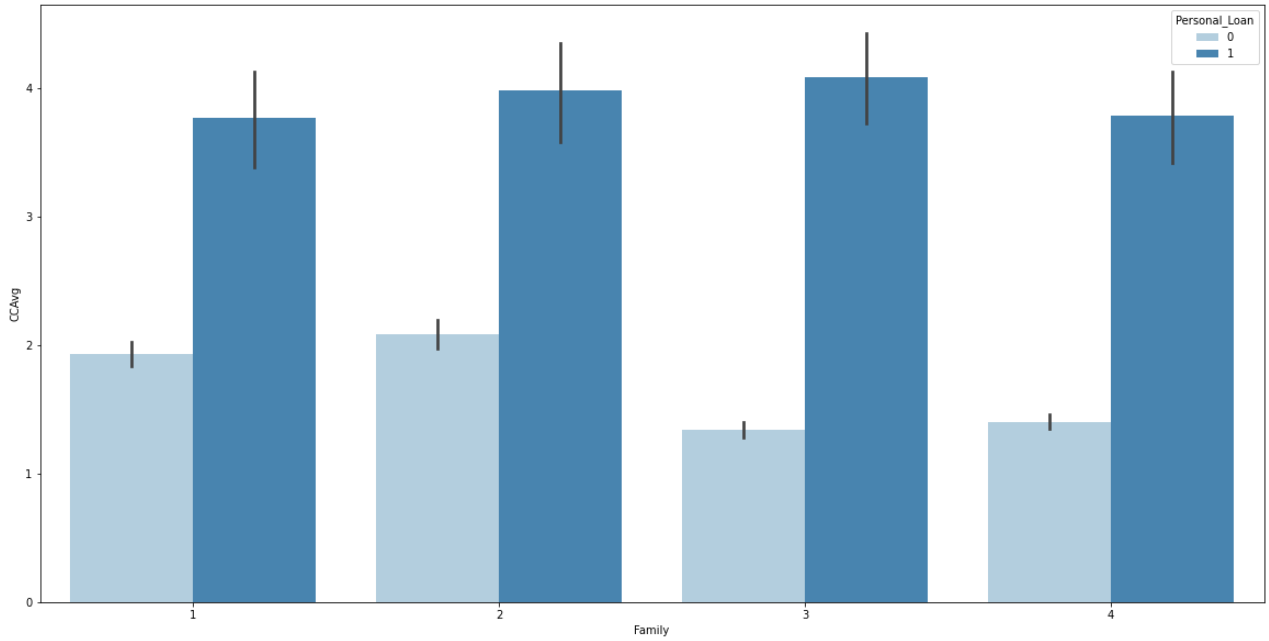


In [520…

```
box_cat_vs_num( df , 'Family' , 'CCAvg' , 'Personal_Loan' , 'Blues')
```

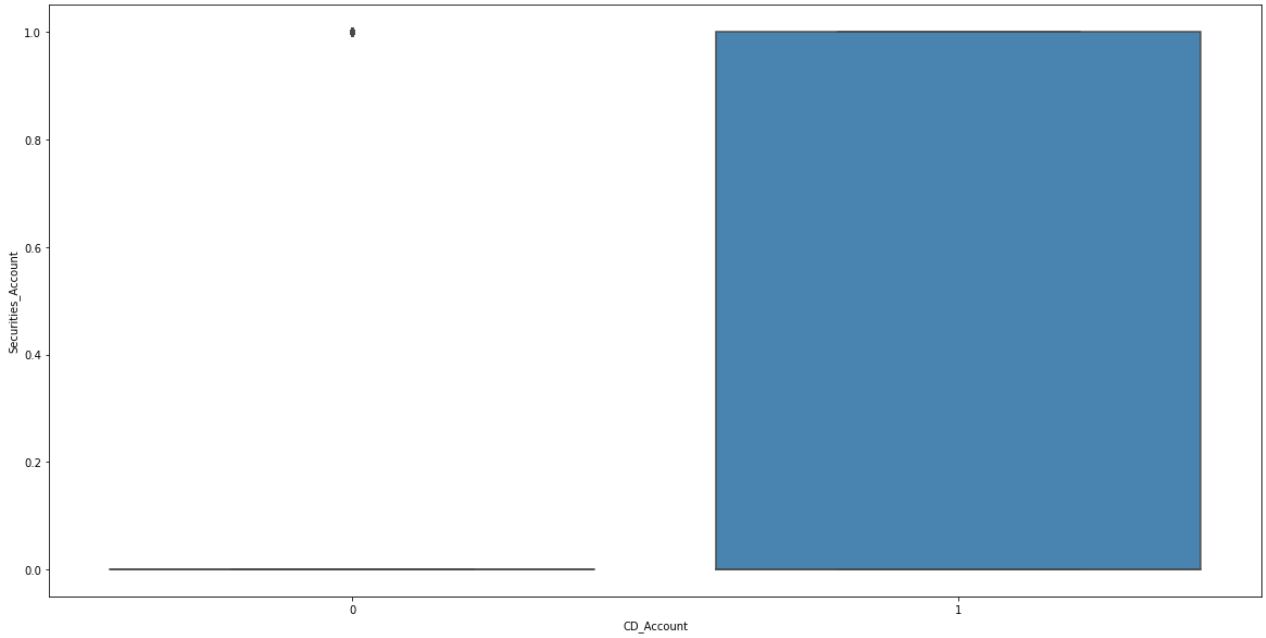


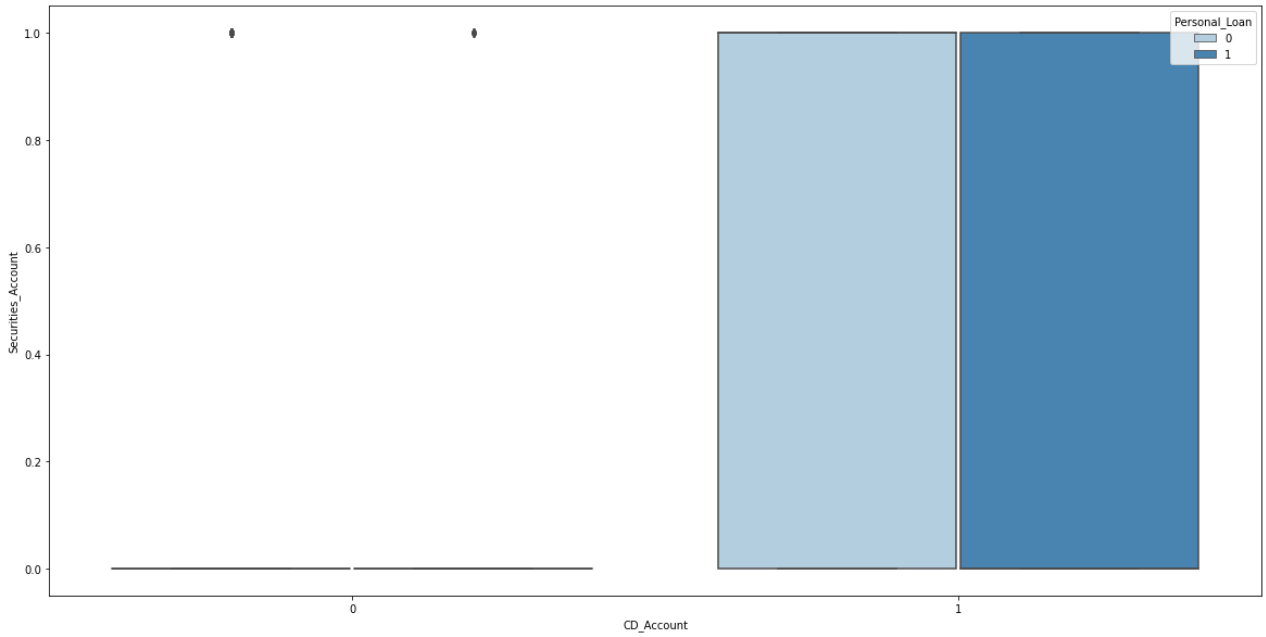


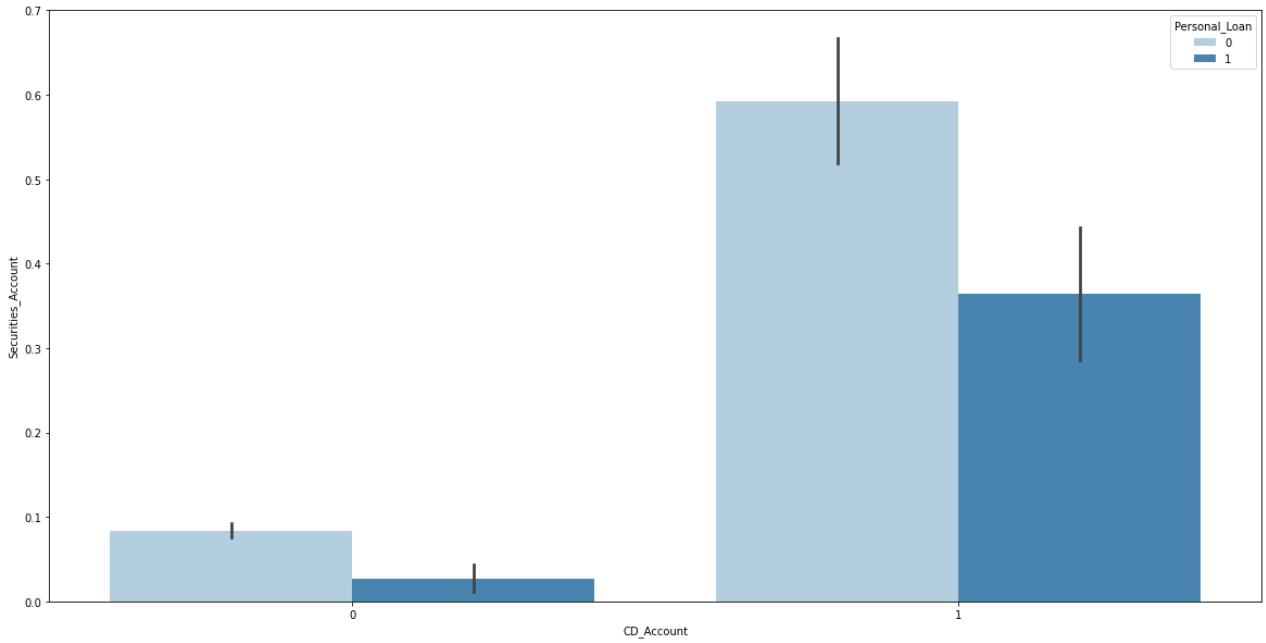


In [520…

```
box_cat_vs_num( df , 'CD_Account' , 'Securities_Account' , 'Personal_Loan' , 'Blues')
```


Securities_Account
1.0
0.8
0.6
0.4
0.2
0.0
0
1
CD_Account


Personal_Loan
0
1
Securities_Account
1.0
0.8
0.6
0.4
0.2
0.0
0
1
CD_Account


Personal_Loan
0
1
Securities_Account
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
1
CD_Account

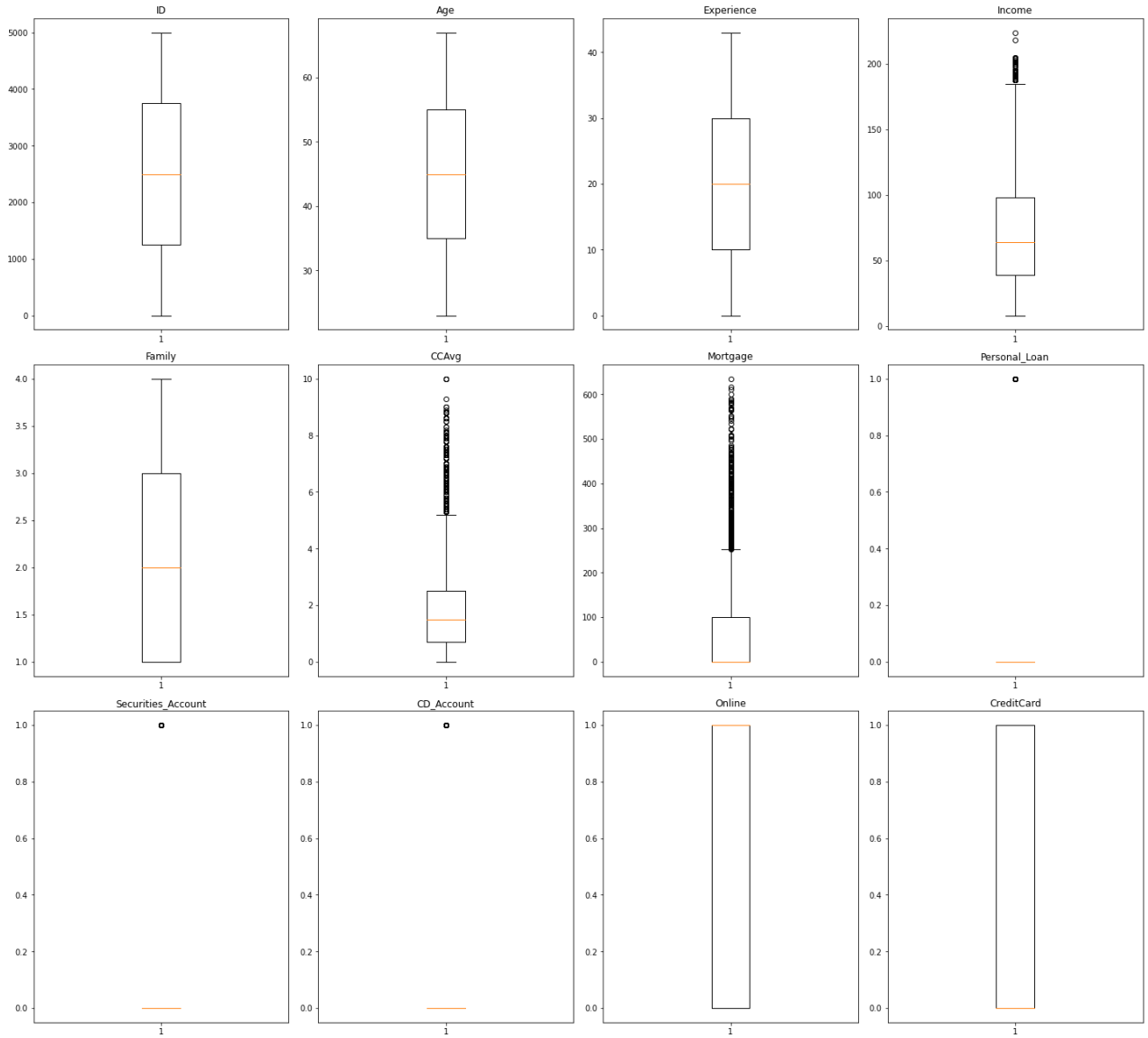
- There is high correlation for customers that have a `CD_Account` and do not have a `Securities_Account` for customer that opened a Personal_Loan.

In [ ]:

Outliers Treatment

In [520…

```
numerical_col = df.select_dtypes(include=np.number).columns.tolist()
plt.figure(figsize=(20,30))

for i, variable in enumerate(numerical_col):
                    plt.subplot(5,4,i+1)
                    plt.boxplot(df[variable],whis=1.5)
                    plt.tight_layout()
                    plt.title(variable)

plt.show()
```



- `Income` , `CC_Avg` and `Mortgage' have upper outliers

In [520…

```
def treat_outliers(df,col):
```

```
    '''
    treats outliers in a varaible
    col: str, name of the numerical varaible
    df: data frame
    col: name of the column
    '''
    Q1=df[col].quantile(0.25) # 25th quantile
    Q3=df[col].quantile(0.75)  # 75th quantile
    IQR=Q3-Q1
    Lower_Whisker = Q1 - 1.5*IQR
    Upper_Whisker = Q3 + 1.5*IQR
    df[col] = np.clip(df[col], Lower_Whisker, Upper_Whisker) # all the values samller t
                                                             # and all the values above
    return df

def treat_outliers_all(df, col_list):
    '''
    treat outlier in all numerical varaibles
    col_list: list of numerical varaibles
    df: data frame
    '''
    for c in col_list:
        df = treat_outliers(df,c)

    return df
```

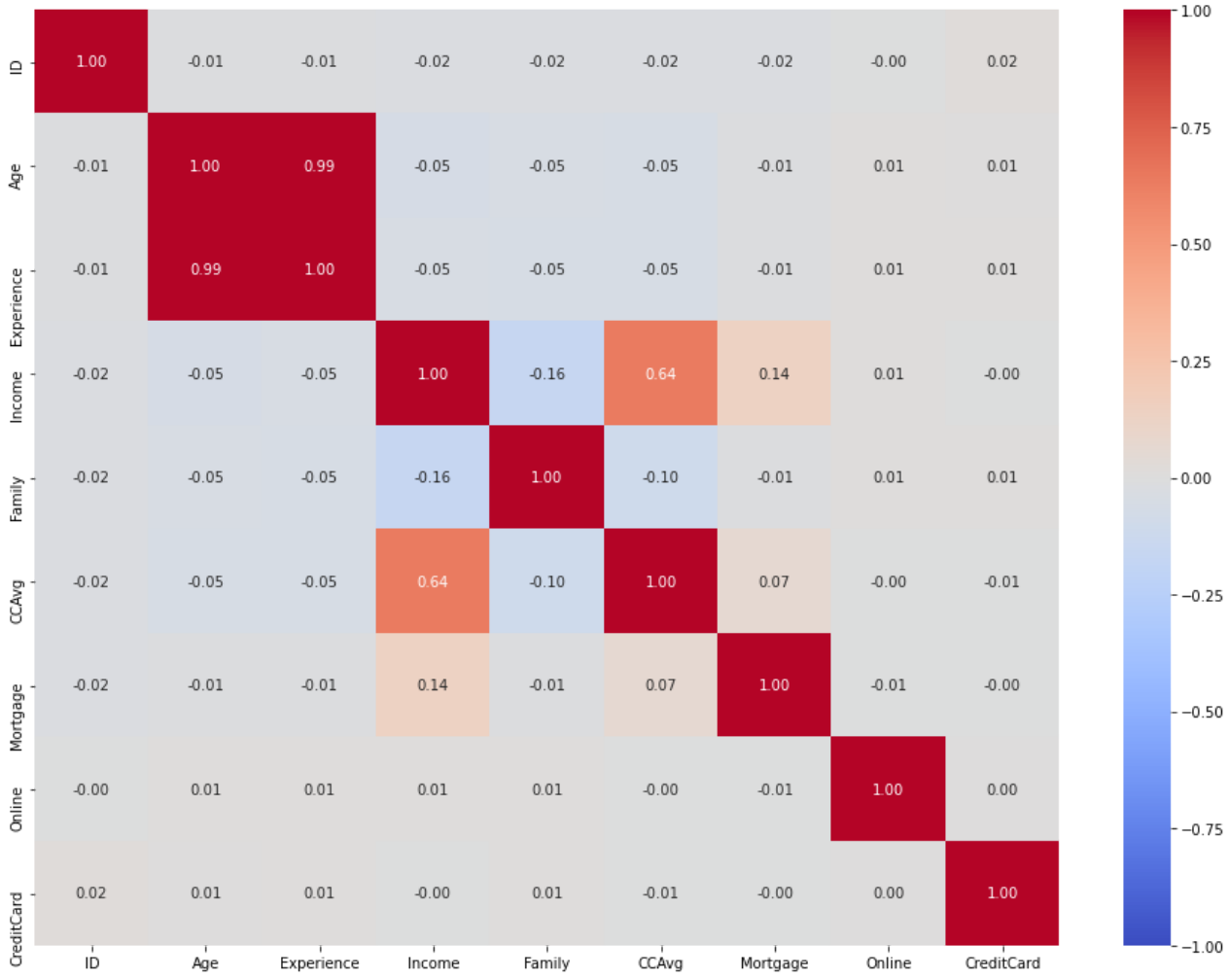
In [520…

```
numerical_col = df.select_dtypes(include=np.number).columns.tolist()# getting list of n


# items to be removed
unwanted= {'Personal_Loan', 'CD_Account' , 'Securities_Account'} # these column have ve

numerical_col = [ele for ele in numerical_col if ele not in unwanted]
df = treat_outliers_all(df,numerical_col)
```

In [ ]:

In [520…

```
corr = df[numerical_col].corr()

# plot the heatmap
plt.figure(figsize=(16,12))
sns.heatmap(corr, annot=True,cmap='coolwarm',vmax=1,vmin=-1,
        fmt=".2f",
        xticklabels=corr.columns,
        yticklabels=corr.columns);
```



- After treating the outliers, we can observe that `Income` has a high correlationsship with `CCAvg` .

In [520…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype   
---  ------              --------------  -----   
 0   ID                  5000 non-null   int64   
 1   Age                 5000 non-null   int64   
 2   Experience          5000 non-null   int64   
 3   Income              5000 non-null   float64 
 4   ZIPCode             5000 non-null   category
 5   Family              5000 non-null   int64   
 6   CCAvg               5000 non-null   float64 
 7   Education           5000 non-null   category
 8   Mortgage            5000 non-null   float64 
 9   Personal_Loan       5000 non-null   int64   
 10  Securities_Account  5000 non-null   int64   
 11  CD_Account          5000 non-null   int64   
 12  Online              5000 non-null   int64   
 13  CreditCard          5000 non-null   int64   
 14  ZIPCodeZone         5000 non-null   object  
dtypes: category(2), float64(3), int64(9), object(1)
memory usage: 546.3+ KB
```

Data Pre-Processing

- Convert non-int and non-bool variables to categorical
  - Family
  - Education
  - ZIPCodeZone
- Create Dummies for Categoral and Boolean variables
- Also remove variables that will not be used for the model:
  - ID
  - ZipCode

In [521…
```
df.head()
```

Out[521…

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_Loan	Securi
0	1	25	1	49.00	91107	4	1.60	1	0.00	0	
1	2	45	19	34.00	90089	3	1.50	1	0.00	0	
2	3	39	15	11.00	94720	1	1.00	1	0.00	0	
3	4	35	9	100.00	94112	1	2.70	2	0.00	0	
4	5	35	8	45.00	91330	4	1.00	2	0.00	0	

- ID has the information related to the transaction activity of a customer and would create a bias in model. So we should drop `ID`

Fixing the data types

- Convert `ZIPCode` , `ZIPCodeZone` , `Family` and `Education` to categorical variables.

In [521…
```
#df["Family"] = df["Family"].astype('category')
#df["Education"] = df["Education"].astype('category')
#df["ZIPCodeZone"] = df["ZIPCodeZone"].astype('category')

categorical_variables = df.select_dtypes(exclude=["number","bool_"]).columns.tolist() #
for colmn in categorical_variables:
    df[colmn] = df[colmn].astype('category')
```

In [521…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   ID                  5000 non-null   int64
 1   Age                 5000 non-null   int64
 2   Experience          5000 non-null   int64
 3   Income              5000 non-null   float64
 4   ZIPCode             5000 non-null   category
 5   Family              5000 non-null   int64
 6   CCAvg               5000 non-null   float64
 7   Education           5000 non-null   category
```

```
 8   Mortgage            5000 non-null   float64
 9   Personal_Loan       5000 non-null   int64
 10  Securities_Account  5000 non-null   int64
 11  CD_Account          5000 non-null   int64
 12  Online              5000 non-null   int64
 13  CreditCard          5000 non-null   int64
 14  ZIPCodeZone         5000 non-null   category
dtypes: category(3), float64(3), int64(9)
memory usage: 512.5 KB
```

In [ ]:

Split Data

- Drop `ID`, since it has information related to the transaction activity of a customer and would create a bias in model.
- Also, drop `ZIPCode`, since we have reducted the list downe to `ZIPCodeZone`.

In [521…
```
df = df.drop(['ID' , 'ZIPCode'], axis=1)
```

In [521…
```
## Defining X and Y variables
X = df.drop(['Personal_Loan'], axis=1)
y = df[['Personal_Loan']]

#Convert categorical variables to dummy variables
X = pd.get_dummies(X, drop_first=True)
#Y['Personal_Loan'] = Y.Personal_Loan
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.30)
```

In [521…
```
# Splitting data into training and test set:
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.3, random_state=1)
print(X_train.shape, X_test.shape)
```

```
(3500, 18) (1500, 18)
```

In [ ]:

Model building - Logistic Regression

In [521…
```
from sklearn import metrics
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression(solver='newton-cg',max_iter=1000,penalty='none',verbose=Tru

# There are several optimizer, we are using optimizer called as 'newton-cg' with max_it
# max_iter indicates number of iteration needed to converge

logreg.fit(X_train, y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:    1.0s finished
```

Out[521…
```
LogisticRegression(max_iter=1000, n_jobs=-1, penalty='none', random_state=0,
                   solver='newton-cg', verbose=True)
```

Model Performances

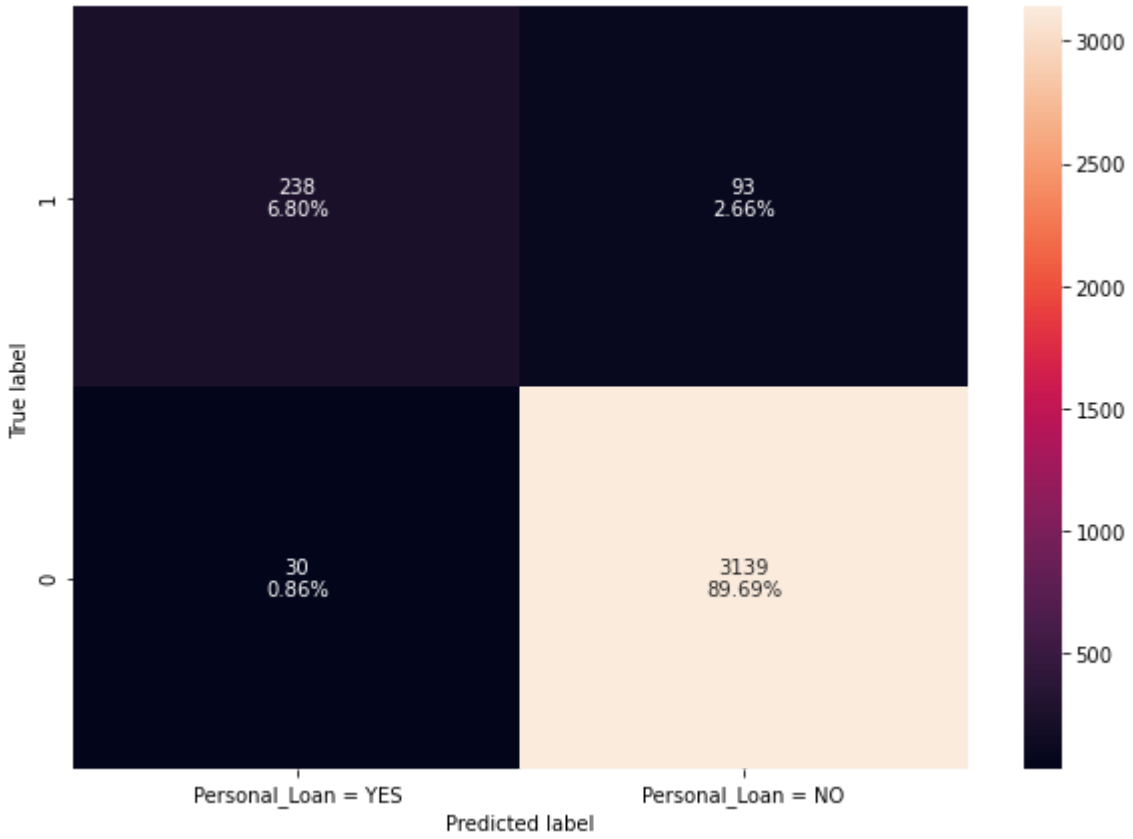
Prediction on training data

In [521…

```
#Predict for train set
pred_train = logreg.predict(X_train)

from sklearn.metrics import classification_report,confusion_matrix
#mat_train = confusion_matrix(y_train,pred_train)


def make_confusion_matrix(y_actual,y_predict,labels=[1, 0]):
    '''
    y_predict: prediction of class
    y_actual : ground truth
    '''
    cm=confusion_matrix( y_actual,y_predict, labels=[1, 0])
    df_cm = pd.DataFrame(cm, index = [i for i in ["1","0"]],
                  columns = [i for i in ['Personal_Loan = YES','Personal_Loan = NO']])
    group_counts = ["{0:0.0f}".format(value) for value in
                cm.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in
                         cm.flatten()/np.sum(cm)]
    labels = [f"{v1}\n{v2}" for v1, v2 in
              zip(group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    plt.figure(figsize = (10,7))
    sns.heatmap(df_cm, annot=labels,fmt='')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [521…

```
make_confusion_matrix(y_train,pred_train)
```



In [521…

```
# Verify confusion matrix results
y_train_noindex = y_train.reset_index( drop = True)['Personal_Loan']
```

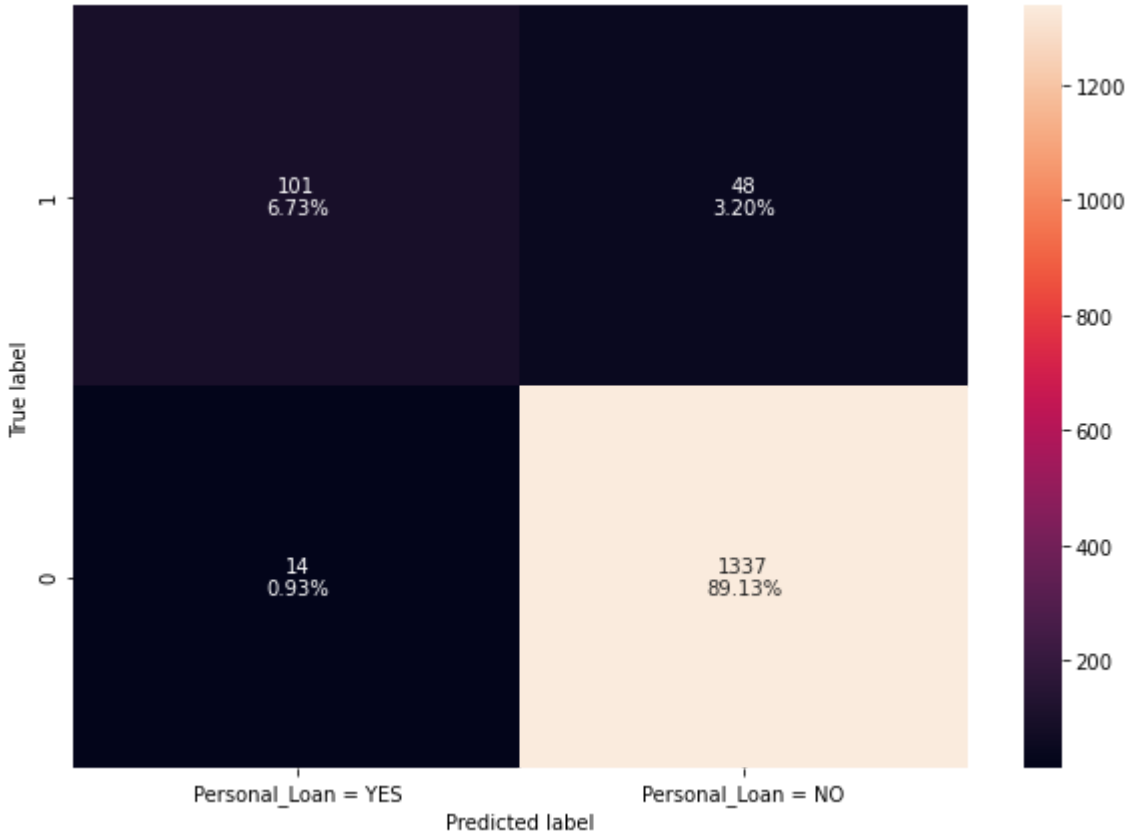
```
df_test = pd.DataFrame( [ pred_train , y_train_noindex ]).T
df_test.columns = [ 'pred_train' , 'y_train'  ]
df_test.groupby ( [ 'pred_train' , 'y_train'  ] ).size()
```

Out[521…
```
pred_train  y_train
0           0          3139
            1            93
1           0            30
            1           238
dtype: int64
```

Prediction on test data

In [522…
```
#Predict for test set
pred_test = logreg.predict(X_test)


print("confusion matrix = \n")
make_confusion_matrix(y_test,pred_test)
```

```
confusion matrix =
```



In [522…
```
# Verify confusion matrix results
y_test_noindex = y_test.reset_index( drop = True)['Personal_Loan']

df_test = pd.DataFrame( [ pred_test , y_test_noindex ]).T
df_test.columns = [ 'pred_test' , 'y_test'  ]
df_test.groupby ( [ 'pred_test' , 'y_test'  ] ).size()
```

Out[522…
```
pred_test  y_test
0          0         1337
           1           48
1          0           14
```

```
           1          101
dtype: int64
```

In [522…
```
#Accuracy with a threhold of 0.5
from sklearn.metrics import accuracy_score
print('Accuracy on train data:',accuracy_score(y_train, pred_train) )
print('Accuracy on test data:',accuracy_score(y_test, pred_test))
```

```
Accuracy on train data: 0.9648571428571429
Accuracy on test data: 0.9586666666666667
```

In [522…
```
# find F1 Score
from sklearn.metrics import f1_score
print('F1 on train data' , f1_score(y_train , pred_train))
print('F1 on test data' , f1_score(y_test , pred_test))
```

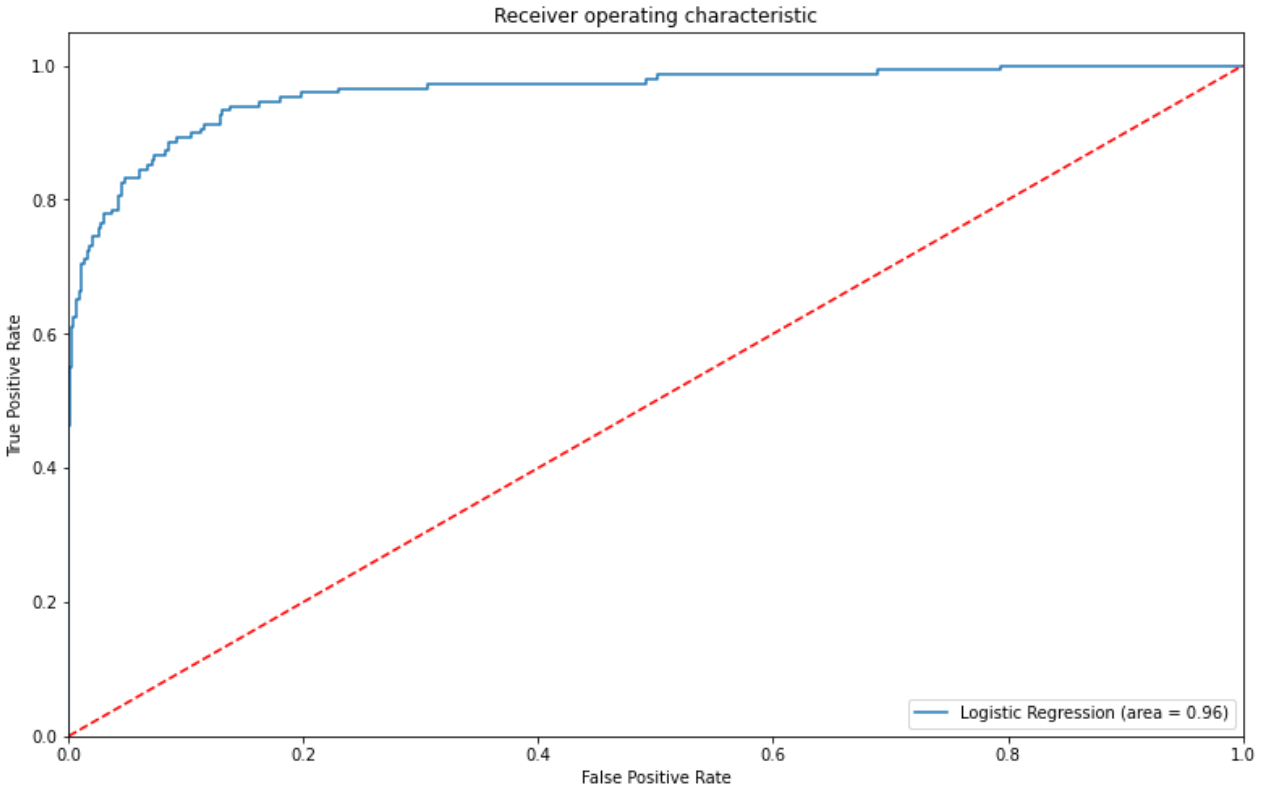
```
F1 on train data 0.7946577629382304
F1 on test data 0.7651515151515152
```

In [522…
```
# find precision recall score
from sklearn import metrics
metrics.precision_recall_fscore_support(y_train , pred_train)
```

Out[522…
```
(array([0.97122525, 0.8880597 ]),
 array([0.99053329, 0.71903323]),
 array([0.98078425, 0.79465776]),
 array([3169,  331], dtype=int64))
```

AUC ROC curve

In [522…
```
#AUC ROC curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve

logit_roc_auc = roc_auc_score(y_test, logreg.predict_proba(X_test)[:,1])
fpr, tpr, thresholds = roc_curve(y_test, logreg.predict_proba(X_test)[:,1])
plt.figure(figsize=(13,8))
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



Optimal threshold

In [522…

```
# The optimal cut off would be where tpr is high and fpr is low
fpr, tpr, thresholds = roc_curve(y_test, logreg.predict_proba(X_test)[:,1])

optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]
print(optimal_threshold)
```

```
0.0459844107610445
```

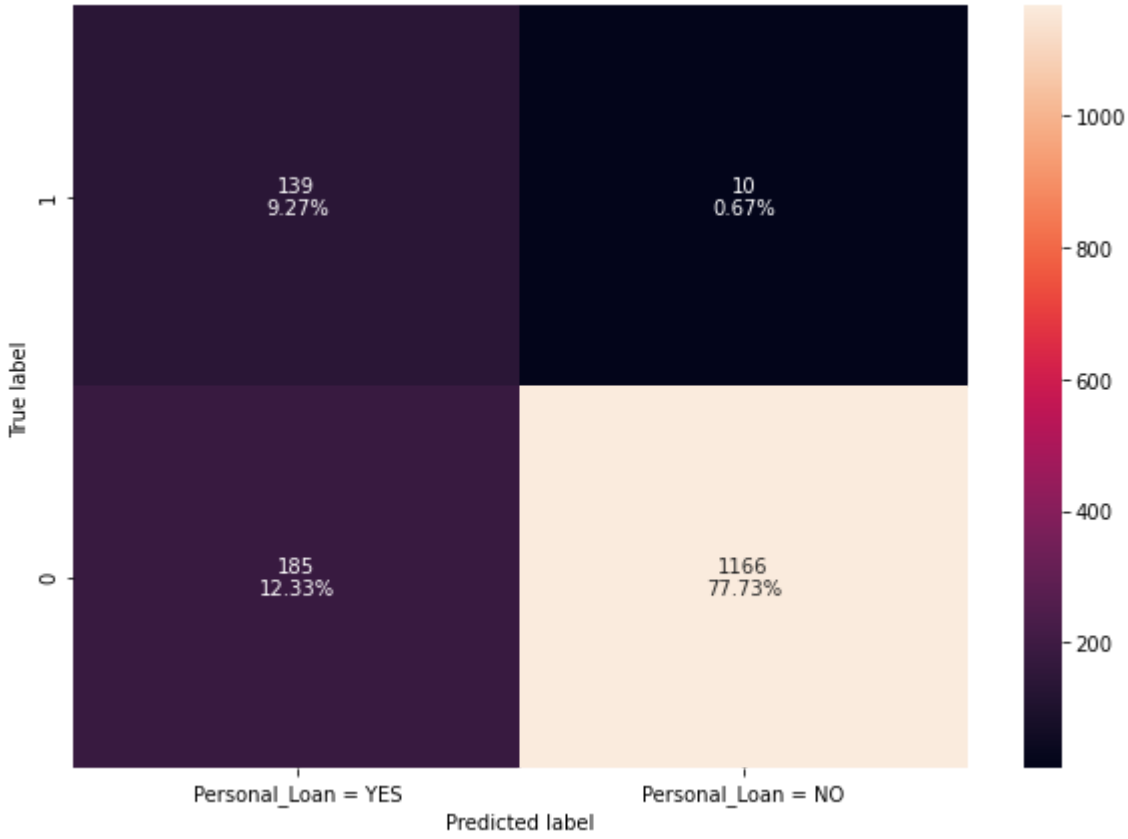
- Use Optimal Threshold to check Model performance

In [522…

```
target_names = ['Personal_Loan = YES', 'Personal_Loan = NO']
y_pred_tr = (logreg.predict_proba(X_train)[:,1]>optimal_threshold).astype(int)
y_pred_ts = (logreg.predict_proba(X_test)[:,1]>optimal_threshold).astype(int)
```

In [522…

```
make_confusion_matrix(y_test,y_pred_ts)
```



In [522…
```
#Accuracy with optimal threhold
from sklearn.metrics import accuracy_score
print('Accuracy on train data:',accuracy_score(y_train, y_pred_tr) )
print('Accuracy on test data:',accuracy_score(y_test, y_pred_ts))
```

```
Accuracy on train data: 0.8668571428571429
Accuracy on test data: 0.87
```

Observation

- After using optimal threshold we see that true positives have increased from 6.8% to 9.2%. and false negatives has decreased from 89.6% to 77.7%

Test assumptions

Check for multicollinearity

- Variance Inflation factor : Measure the inflation in the variances of the regression coefficients estimates due to collinearities that exist among the predictors. It is a measure of how much the variance of the estimated regression coefficient βk is “inflated”by the existence of correlation among the predictor variables in the model.

In [523…
```
# dataframe with numerical column only
num_feature_set = X.copy()
from statsmodels.tools.tools import add_constant
num_feature_set = add_constant(num_feature_set)
```

In [523…
```
vif_series1 = pd.Series([variance_inflation_factor(num_feature_set.values,i) for i in r
print('Series before feature selection: \n\n{}\n'.format(vif_series1))
```

```
Series before feature selection:

const                465.18
Age                   93.21
Experience            93.10
Income                 1.81
Family                 1.05
CCAvg                  1.70
Mortgage               1.02
Securities_Account     1.14
CD_Account             1.33
Online                 1.04
CreditCard             1.11
Education_2            1.29
Education_3            1.33
ZIPCodeZone_91         1.60
ZIPCodeZone_92         1.93
ZIPCodeZone_93         1.46
ZIPCodeZone_94         2.19
ZIPCodeZone_95         1.81
ZIPCodeZone_96         1.05
dtype: float64
```

- We observe that Age, and Experience have high multicollinearity
- As detected in the EDA, we noticeded a very high correlation between these variables. We will go ahead and drop the higher VIF value `Age` .

In [523…
```
num_feature_set = num_feature_set.drop( [ 'Age' ] ,axis=1)
```

- we can check for VIF again

In [523…
```
vif_series1 = pd.Series([variance_inflation_factor(num_feature_set.values,i) for i in r
print('Series before feature selection: \n\n{}\n'.format(vif_series1))
```

```
Series before feature selection:

const                23.88
Experience            1.01
Income                1.81
Family                1.05
CCAvg                 1.70
Mortgage              1.02
Securities_Account    1.14
CD_Account            1.33
Online                1.04
CreditCard            1.11
Education_2           1.27
Education_3           1.25
ZIPCodeZone_91        1.60
ZIPCodeZone_92        1.93
ZIPCodeZone_93        1.46
ZIPCodeZone_94        2.19
ZIPCodeZone_95        1.81
ZIPCodeZone_96        1.05
dtype: float64
```

Model Building - with Predictor data set

- Now that we have built our predictor set `num_feature_set` . we can re-evaluate the Logistic Regression model again.

Split `num_feature_set` into training and test set

In [523…
```
X_train, X_test, y_train, y_test = train_test_split(num_feature_set, Y, test_size=0.30)

print("Shape of y:", y_train.shape, " &&Shape of X_selected_lsvc:", X_train.shape)
print("y values:",y_train.head())
```

```
Shape of y: (3500, 1)  &&Shape of X_selected_lsvc: (3500, 18)
y values:       Personal_Loan
3952              0
4799              0
4306              0
2927              0
1674              1
```

Building Logistic Regression model from statsmodels

In [523…
```
import statsmodels.api as sm
logit = sm.Logit(y_train , X_train )
lg = logit.fit()

#lg = sm.OLS(y, X.astype(float)).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.117731
         Iterations 10
```

Logit Regression Summary

In [523…
```
print(lg.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:          Personal_Loan   No. Observations:                 3500
Model:                          Logit   Df Residuals:                     3482
Method:                           MLE   Df Model:                           17
Date:                Sat, 08 May 2021   Pseudo R-squ.:                  0.6292
Time:                        04:31:00   Log-Likelihood:                -412.06
converged:                       True   LL-Null:                       -1111.2
Covariance Type:            nonrobust   LLR p-value:                3.645e-287
==============================================================================
                         coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const                -13.1444      0.739    -17.780      0.000     -14.593     -11.695
Experience             0.0066      0.008      0.808      0.419      -0.009       0.023
Income                 0.0583      0.004     16.401      0.000       0.051       0.065
Family                 0.5781      0.090      6.402      0.000       0.401       0.755
CCAvg                  0.4374      0.069      6.381      0.000       0.303       0.572
Mortgage               0.0010      0.001      1.055      0.292      -0.001       0.003
Securities_Account    -0.8714      0.360     -2.420      0.016      -1.577      -0.166
CD_Account             3.7311      0.416      8.972      0.000       2.916       4.546
Online                -0.8127      0.200     -4.072      0.000      -1.204      -0.422
CreditCard            -1.1725      0.257     -4.568      0.000      -1.676      -0.669
Education_2            3.9897      0.320     12.474      0.000       3.363       4.617
Education_3            3.9409      0.309     12.754      0.000       3.335       4.546
ZIPCodeZone_91        -0.4475      0.343     -1.303      0.193      -1.120       0.226
ZIPCodeZone_92        -0.0428      0.305     -0.140      0.888      -0.641       0.555
ZIPCodeZone_93        -0.1294      0.394     -0.328      0.743      -0.902       0.643
ZIPCodeZone_94        -0.4193      0.291     -1.441      0.150      -0.990       0.151
ZIPCodeZone_95        -0.2918      0.316     -0.923      0.356      -0.911       0.328
```

```
ZIPCodeZone_96        -2.2505      3.061     -0.735      0.462      -8.249       3.748
================================================================================================

Possibly complete quasi-separation: A fraction 0.12 of observations can be
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.
```

- Checking for P-Value >.000. We see that `Experience` , `Mortgage` and `Securities_Account` have little significance to the model. We will cleanup and re-run the stats model.
- Dummie variables can be ignored for now.

In [523…
```
num_feature_set = num_feature_set.drop( [ 'Experience' , 'Mortgage' , 'Securities_Accou
```

In [523…
```
vif_series1 = pd.Series([variance_inflation_factor(num_feature_set.values,i) for i in r
print('Series before feature selection: \n\n{}\n'.format(vif_series1))
```

```
Series before feature selection:

const            19.30
Income            1.78
Family            1.04
CCAvg             1.69
CD_Account        1.17
Online            1.04
CreditCard        1.09
Education_2       1.27
Education_3       1.25
ZIPCodeZone_91    1.60
ZIPCodeZone_92    1.93
ZIPCodeZone_93    1.46
ZIPCodeZone_94    2.19
ZIPCodeZone_95    1.81
ZIPCodeZone_96    1.05
dtype: float64
```

In [523…
```
X_train, X_test, y_train, y_test = train_test_split(num_feature_set, Y, test_size=0.30)

logit = sm.Logit(y_train , X_train )
lg = logit.fit()

print(lg.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.117230
         Iterations 10
                           Logit Regression Results
==============================================================================
Dep. Variable:          Personal_Loan   No. Observations:                 3500
Model:                          Logit   Df Residuals:                     3485
Method:                           MLE   Df Model:                           14
Date:                Sat, 08 May 2021   Pseudo R-squ.:                  0.6359
Time:                        04:31:00   Log-Likelihood:                -410.30
converged:                       True   LL-Null:                       -1126.8
Covariance Type:            nonrobust   LLR p-value:                1.326e-297
==============================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const            -13.5516      0.742    -18.269      0.000     -15.005     -12.098
Income             0.0615      0.004     16.983      0.000       0.054       0.069
Family             0.6307      0.092      6.828      0.000       0.450       0.812
```

```
CCAvg              0.3857      0.068      5.640      0.000       0.252       0.520
CD_Account         3.0559      0.370      8.262      0.000       2.331       3.781
Online            -0.6587      0.198     -3.318      0.001      -1.048      -0.270
CreditCard        -1.0955      0.253     -4.328      0.000      -1.592      -0.599
Education_2        3.9603      0.322     12.314      0.000       3.330       4.591
Education_3        3.9723      0.310     12.808      0.000       3.364       4.580
ZIPCodeZone_91    -0.3957      0.356     -1.111      0.267      -1.094       0.303
ZIPCodeZone_92     0.1027      0.303      0.339      0.735      -0.491       0.697
ZIPCodeZone_93    -0.5009      0.416     -1.203      0.229      -1.317       0.315
ZIPCodeZone_94    -0.2134      0.294     -0.726      0.468      -0.789       0.362
ZIPCodeZone_95    -0.0115      0.317     -0.036      0.971      -0.633       0.610
ZIPCodeZone_96    -2.4224      3.685     -0.657      0.511      -9.645       4.800
================================================================================

Possibly complete quasi-separation: A fraction 0.12 of observations can be
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.
```

- With the exception of dummie variables, all features have a P_Value of .000

Calculate the odds ratio from the coef using the formula odds ratio=exp(coef)

Calculate the probability from the odds ratio using the formula probability = odds / (1+odds)

In [524…

```
#Calculate Odds Ratio, probability
#create a data frame to collate Odds ratio, probability and p-value of the coef

lgcoef = pd.DataFrame(lg.params, columns=['coef'])
lgcoef.loc[:, "Odds_ratio"] = np.exp(lgcoef.coef)
lgcoef['probability'] = lgcoef['Odds_ratio']/(1+lgcoef['Odds_ratio'])
lgcoef['pval']=lg.pvalues
pd.options.display.float_format = '{:.2f}'.format
```

In [524…

```
# FIlter by significant p-value (pval <0.005) and sort descending by Odds ratio

lgcoef = lgcoef.sort_values(by="Odds_ratio", ascending=False)
pval_filter = lgcoef['pval']<=0.005
lgcoef[pval_filter]
```

Out[524…

	coef	Odds_ratio	probability	pval
Education_3	3.97	53.11	0.98	0.00
Education_2	3.96	52.47	0.98	0.00
CD_Account	3.06	21.24	0.96	0.00
Family	0.63	1.88	0.65	0.00
CCAvg	0.39	1.47	0.60	0.00
Income	0.06	1.06	0.52	0.00
Online	-0.66	0.52	0.34	0.00
CreditCard	-1.10	0.33	0.25	0.00
const	-13.55	0.00	0.00	0.00

Most significant variable

In [524…
```
# we are looking are overall significant varaible

pval_filter = lgcoef['pval']<=0.0001
imp_vars = lgcoef[pval_filter].index.tolist()

# we are going to get overall varaibles (un-one-hot encoded varables) from categorical

sig_var = []
for col in imp_vars:
    if '_' in col:
        first_part = col.split('_')[0]
        for c in df.columns:
            if first_part in c and c not in sig_var :
                sig_var.append(c)


start = '\033[1m'
end = '\033[95m'
print('Most significant varaibles category wise are :\n',lgcoef[pval_filter].index.toli
print('*'*120)

print(start+'Most overall significant varaibles  are '+end,':\n',sig_var)
```

```
Most significant varaibles category wise are :
 ['Education_3', 'Education_2', 'CD_Account', 'Family', 'CCAvg', 'Income', 'CreditCard',
'const']
************************************************************************************
********************************
Most overall significant varaibles  are  :
 ['Education', 'CD_Account']
```

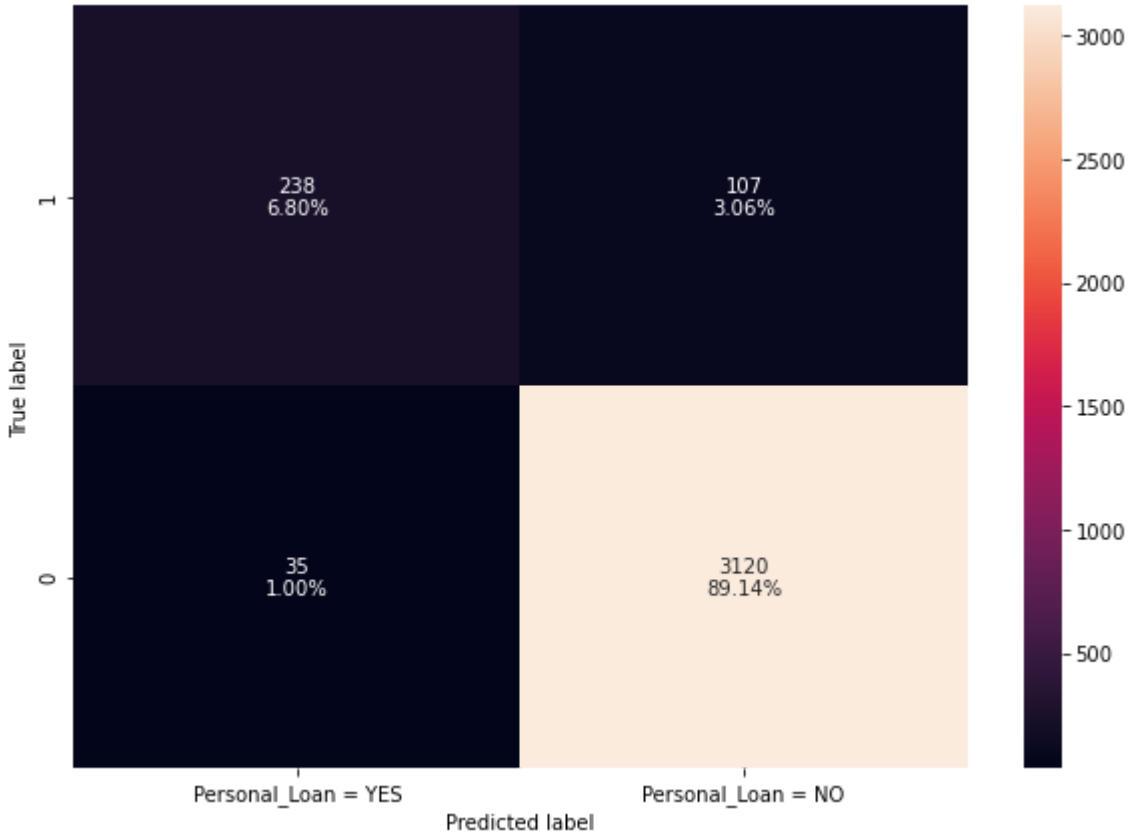
Prediction of the model

Prediction on Train data

In [524…
```
pred_train = lg.predict(X_train)
pred_train = np.round(pred_train)
```

In [524…
```
print("confusion matrix = \n")
make_confusion_matrix(y_train,pred_train )
```

```
confusion matrix =
```



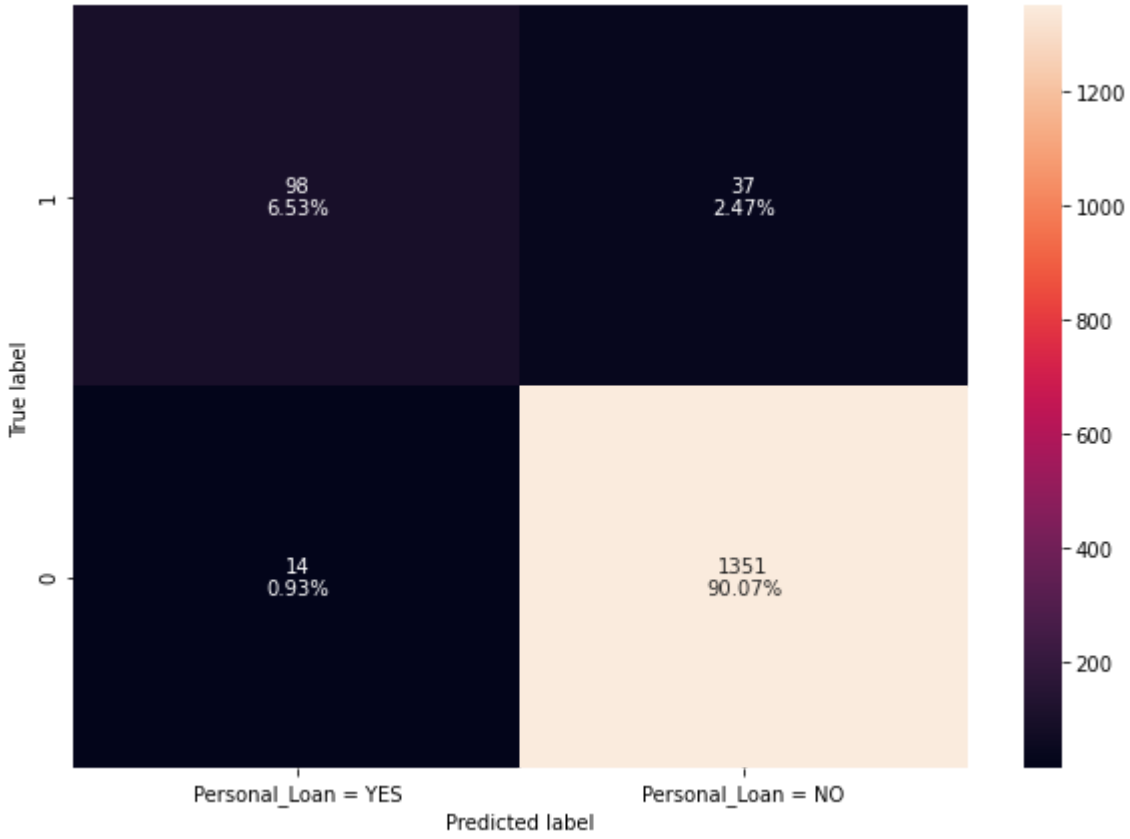
In [ ]:

Prediction on Test data

In [524…

```
pred_ts = lg.predict(X_test)#predict(X_train)
pred_ts = np.round(pred_ts)
# mat_tst = confusion_matrix(y_test,np.round(pred_ts))

print("confusion matrix = \n")
make_confusion_matrix(y_test,pred_ts )
```

```
confusion matrix =
```



In [524…

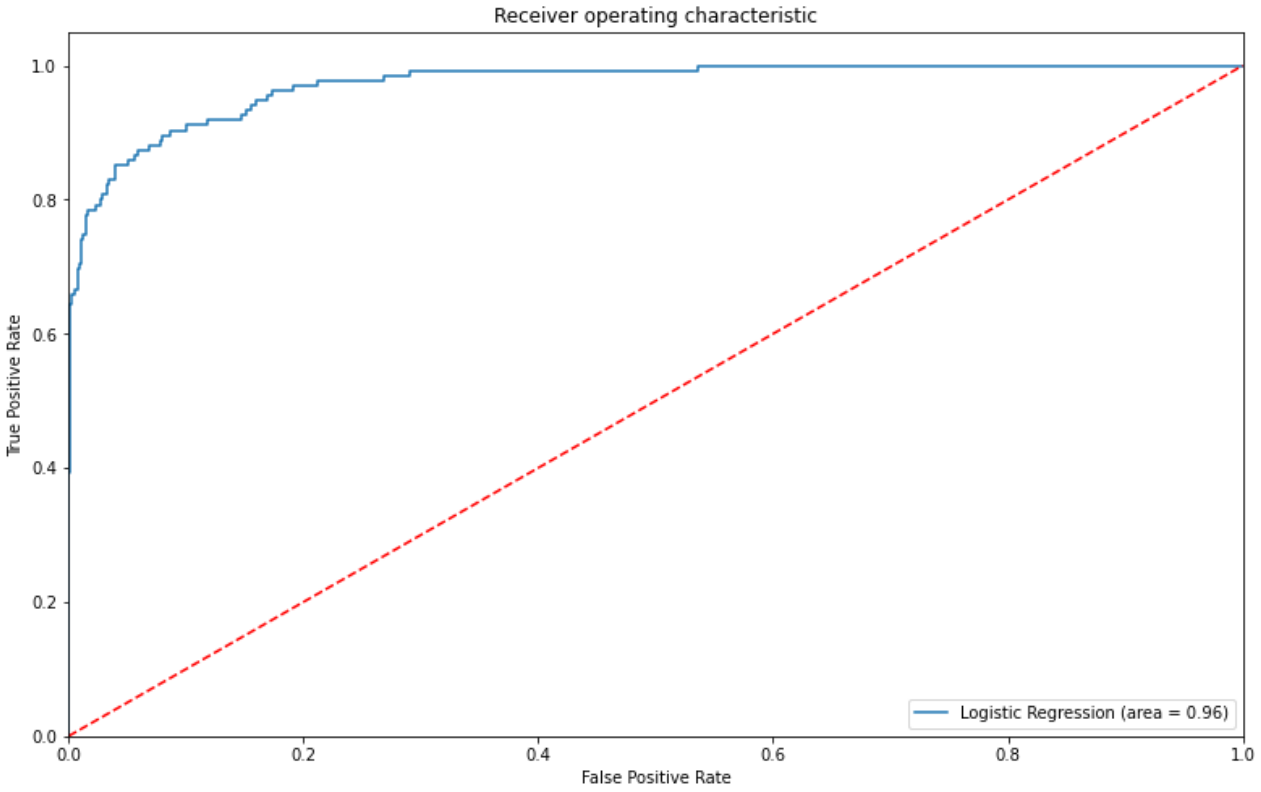
```
#Accuracy with a threhold of 0.5
print('Accuracy on train data:',accuracy_score(y_train, pred_train) )
print('Accuracy on test data:',accuracy_score(y_test, pred_ts))
```

```
Accuracy on train data: 0.9594285714285714
Accuracy on test data: 0.966
```

AUC ROC Curve

In [524…

```
fpr, tpr, thresholds = roc_curve(y_test, lg.predict(X_test))
plt.figure(figsize=(13,8))
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



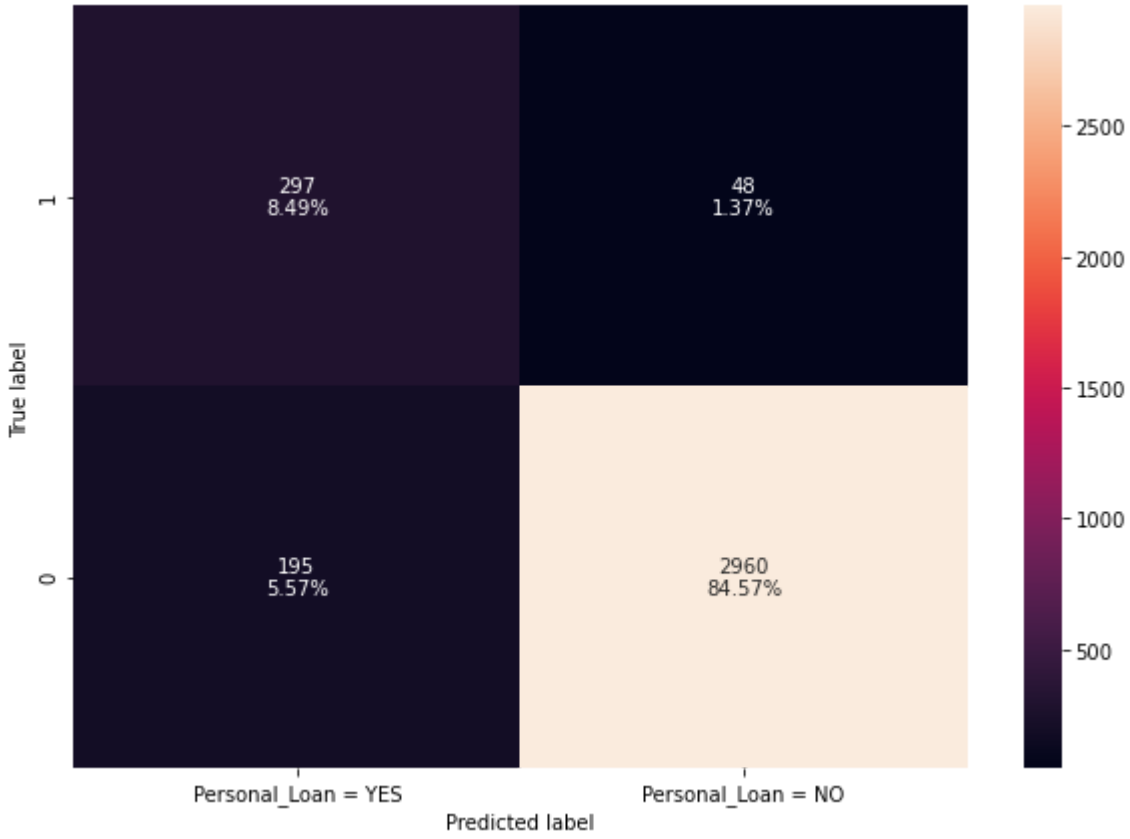
Choosing Optimal threshold

In [524…
```
pred_train = lg.predict(X_train)
# The optimal cut off would be where tpr is high and fpr is low
fpr, tpr, thresholds = roc_curve(y_train, pred_train)

optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]
print(optimal_threshold)
```

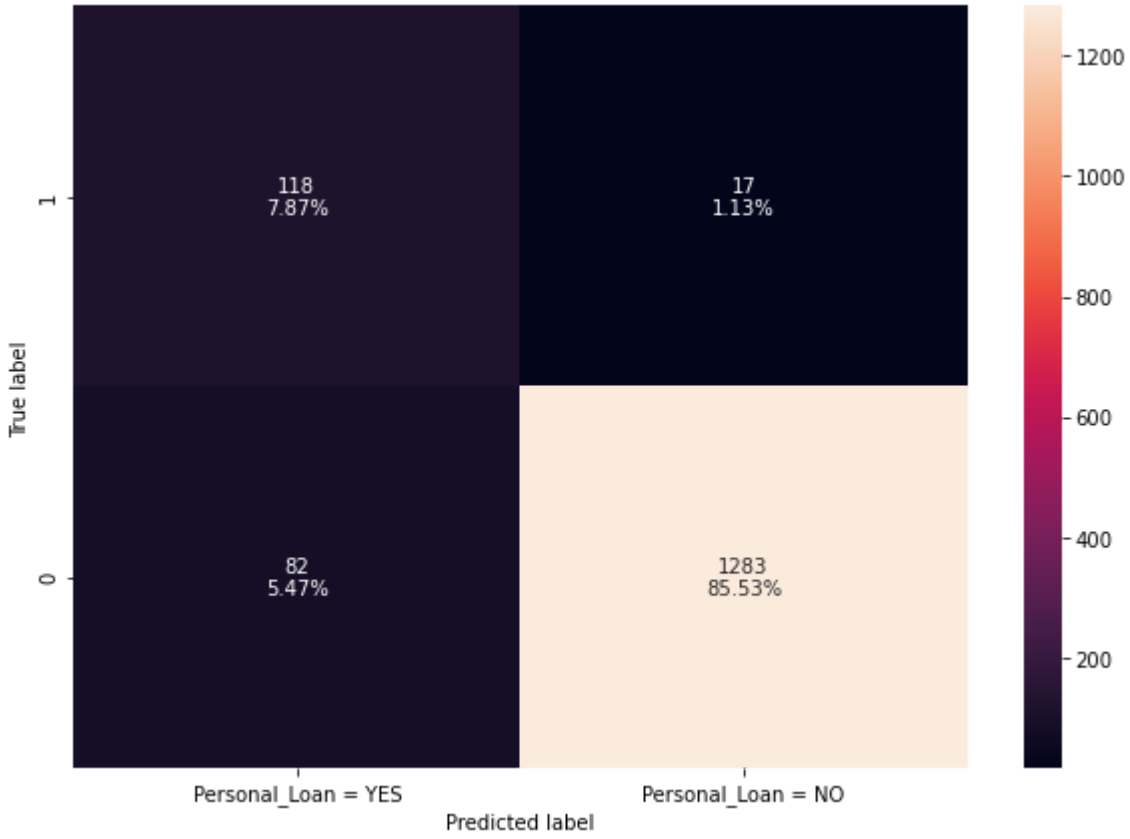
```
0.17208019125159668
```

In [524…
```
target_names = ['Personal_Loan = NO', 'Personal_Loan = YES']
y_pred_tr = (lg.predict(X_train)>optimal_threshold).astype(int)
y_pred_ts = (lg.predict(X_test)>optimal_threshold).astype(int)
```

In [525…
```
make_confusion_matrix(y_train,y_pred_tr )
```



In [525…
```
make_confusion_matrix(y_test,y_pred_ts)
```



In [525…
```
#Accuracy with optimal threshold
print('Accuracy on train data:',accuracy_score(y_train,y_pred_tr) )
print('Accuracy on test data:',accuracy_score(y_test,y_pred_ts))
```

```
Accuracy on train data: 0.9305714285714286
Accuracy on test data: 0.934
```

Observation

- After choosing optimal threshold , true positives has increased from 6.8% to 8.9% while false negative has decreased from 89.6% to 83.4%
- The accuracy on test data is 92% after removing multicollinearity and choosing optimal threshold. This is more reliable as multi collinearity has been removed from the data.
- AUC is 0.96

In [525…
```
import pandas as pd
comparison_frame = pd.DataFrame({'Model':[
    'Initial Logistic Regression Model with sklearn',
    'optimal threshold - Logistic Regression Model with sklearn',
    'Logistic Regression with feature elimination one by one',
    'optimal threshold - Logistic Regression with feature elimination one by one',
                                        ],
    'Train_Accuracy':[0.964 , 0.866 , 0.961 , 0.909],
    'Test_Accuracy':[0.958 , 0.87 , 0.962 , 0.894 ] })

comparison_frame
```

Out[525…

	Model	Train_Accuracy	Test_Accuracy
0	Initial Logistic Regression Model with sklearn	0.96	0.96
1	optimal threshold - Logistic Regression Model ...	0.87	0.87
2	Logistic Regression with feature elimination o...	0.96	0.96
3	optimal threshold - Logistic Regression with f...	0.91	0.89

Conclusions

- `Initial Logistic Regression Model with sklearn` and `Logistic Regression with feature elimination` both have very high accuracy at 0.96 on train and testvalues.

In [ ]:
```
```

Model building - Decision Tree

1. Data preparation
2. Partition the data into train and test set.
3. Built a CART model on the train data.
4. Tune the model and prune the tree, if required.
5. Test the data on test set.

Data preparation

In [525…
```
dummy_data = pd.get_dummies(df, columns=['ZIPCodeZone','Education','Family','Securities
dummy_data.head()
```

Out[525…

	Age	Experience	Income	CCAvg	Mortgage	Personal_Loan	ZIPCodeZone_91	ZIPCodeZone_92	ZIP
0	25	1	49.00	1.60	0.00	0	1	0	
1	45	19	34.00	1.50	0.00	0	0	0	
2	39	15	11.00	1.00	0.00	0	0	0	
3	35	9	100.00	2.70	0.00	0	0	0	
4	35	8	45.00	1.00	0.00	0	1	0	

In [525…

```
column_names = list(dummy_data.columns)
column_names.remove('Personal_Loan')                    # Keep only names of features
feature_names = column_names
print(feature_names)
```

```
['Age', 'Experience', 'Income', 'CCAvg', 'Mortgage', 'ZIPCodeZone_91', 'ZIPCodeZone_92',
'ZIPCodeZone_93', 'ZIPCodeZone_94', 'ZIPCodeZone_95', 'ZIPCodeZone_96', 'Education_2',
'Education_3', 'Family_2', 'Family_3', 'Family_4', 'Securities_Account_1', 'CD_Account_
1', 'Online_1', 'CreditCard_1']
```

In [ ]:

Split data

In [525…

```
X = dummy_data.drop('Personal_Loan',axis=1)
y = dummy_data['Personal_Loan'].astype('int64')
# converting target to integers - since some functions might not work with bool type
```

In [525…

```
# Splitting data into training and test set:
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.3, random_state=1)
print(X_train.shape, X_test.shape)
```

```
(3500, 20) (1500, 20)
```

Build Decision Tree Model (raw)

- We will build our model using the DecisionTreeClassifier function. Using default 'gini' criteria to split.
- In this case, we can pass a dictionary {0:0.10,1:0.90} to the model to specify the weight of each class and the decision tree will give more weightage to class 1.

In [525…

```
model = DecisionTreeClassifier(criterion='gini',class_weight={0:0.10,1:0.90},random_sta

model.fit(X_train, y_train)
```

Out[525…

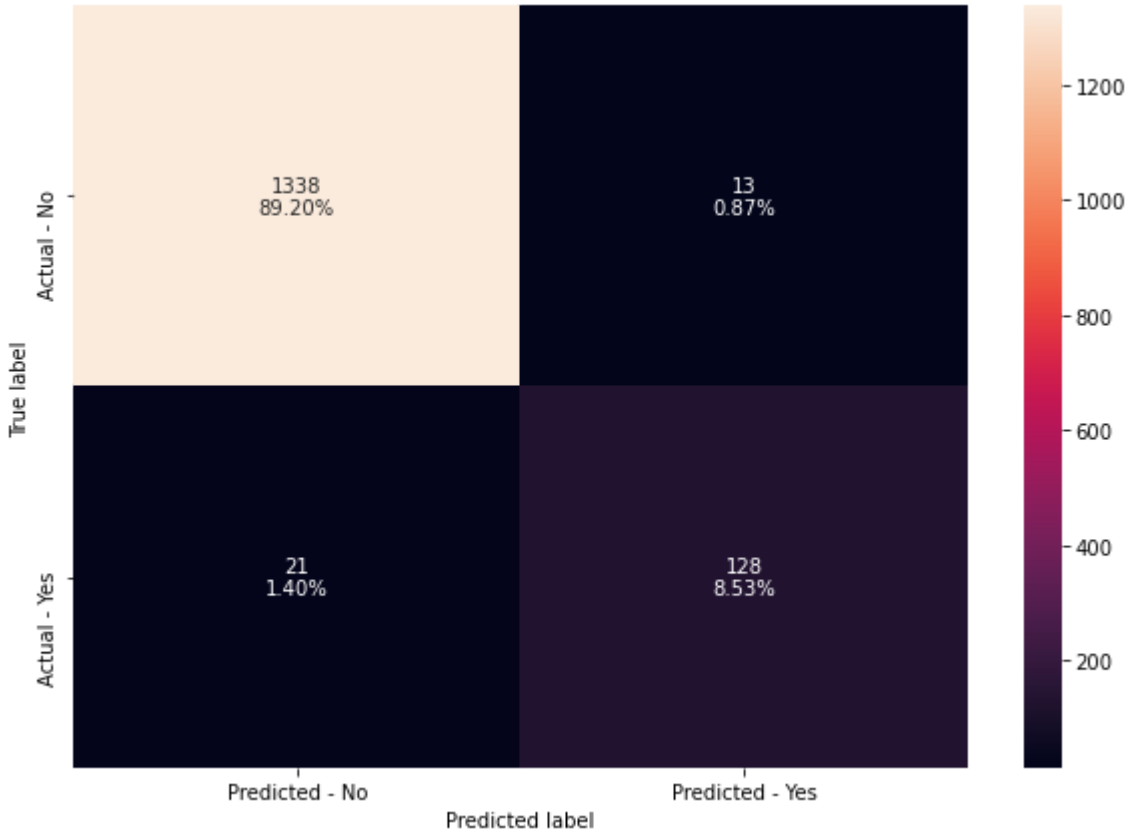
```
DecisionTreeClassifier(class_weight={0: 0.1, 1: 0.9}, random_state=1)
```

In [525…

```
def make_confusion_matrix(model,y_actual,labels=[1, 0]):
    '''
    model : classifier to predict values of X
    y_actual : ground truth

    '''
    y_predict = model.predict(X_test)
```

```
    cm=metrics.confusion_matrix( y_actual, y_predict, labels=[0, 1])
    df_cm = pd.DataFrame(cm, index = [i for i in ["Actual - No","Actual - Yes"]],
                         columns = [i for i in ['Predicted - No','Predicted - Yes']])
    group_counts = ["{0:0.0f}".format(value) for value in
                    cm.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in
                         cm.flatten()/np.sum(cm)]
    labels = [f"{v1}\n{v2}" for v1, v2 in
              zip(group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    plt.figure(figsize = (10,7))
    sns.heatmap(df_cm, annot=labels,fmt='')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [526…
```
make_confusion_matrix(model,y_test)
```



In [526…
```
y_train.value_counts(1)
```

Out[526…
```
0   0.91
1   0.09
Name: Personal_Loan, dtype: float64
```

We only have 10% of positive classes, so if our model marks each sample as negative, then also we'll get 90% accuracy, hence accuracy is not a good metric to evaluate here.

Insights:

- **True Positives:**
  - Reality: A customer converted to Asset Customer.

    - Model predicted: The customer will convert to Asses Customer.
    - Outcome: The model is good.
- **True Negatives:**
    - Reality: A customer did NOT convert to Asset Customer.
    - Model predicted: The customer will NOT converted to Asset Customer.
    - Outcome: The business is unaffected.
- **False Positives:**
    - Reality: A customer did NOT convert to Asset Customer.
    - Model predicted: The customer will convert to Asses Customer.
    - Outcome: The team which is targeting the potential customers will be wasting their resources on the customers who will not convert to Asses Customer.
- **False Negatives:**
    - Reality: A customer converted to Asset Customer.
    - Model predicted: The customer will NOT converted to Asset Customer.
    - Outcome: The potential customer is missed by the marketing team, the team could have offered the potential customer to convert to Asset Account.

- In this case, not being able to identify a potential customer conversion is the biggest loss we can face. Hence, recall is the right metric to check the performance of the model.

In [526…
```
##  Function to calculate recall score
def get_recall_score(model):
    '''
    model : classifier to predict values of X

    '''
    pred_train = model.predict(X_train)
    pred_test = model.predict(X_test)
    print("Recall on training set : ",metrics.recall_score(y_train,pred_train))
    print("Recall on test set : ",metrics.recall_score(y_test,pred_test))
```

In [526…
```
get_recall_score(model)
```

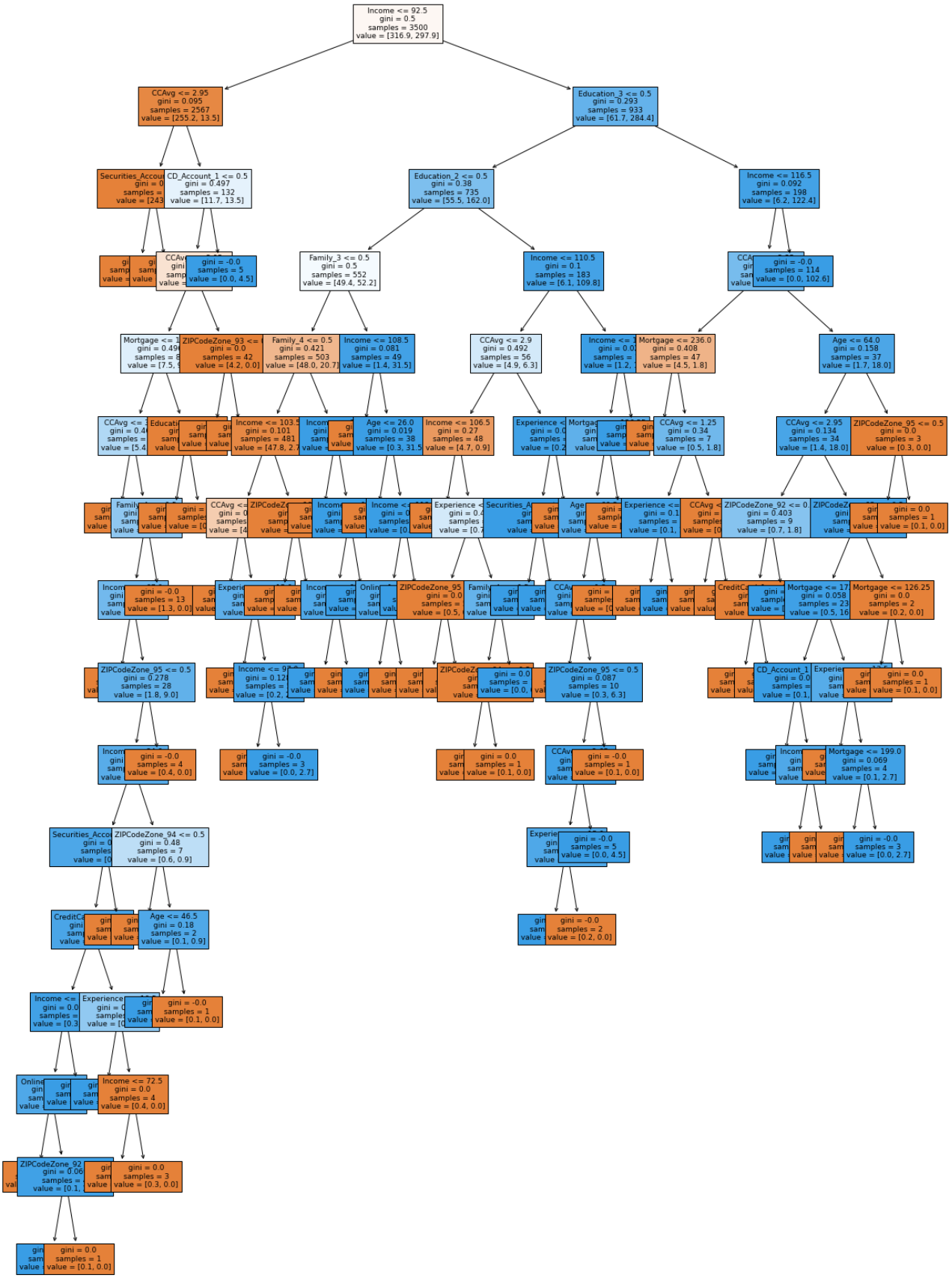
```
Recall on training set :  1.0
Recall on test set :  0.8590604026845637
```

- There is a some disparity in performance of model on training set and test set, which suggests that the model is overfiiting.

Visualizing the Decision Tree (raw)

In [526…
```
plt.figure(figsize=(20,30))
out = tree.plot_tree(model,feature_names=feature_names,filled=True,fontsize=9,node_ids=
#below code will add arrows to the decision tree split if they are missing
for o in out:
     arrow = o.arrow_patch
     if arrow is not None:
        arrow.set_edgecolor('black')
```

```
        arrow.set_linewidth(1)
plt.show()
```



In [526…

```
# Text report showing the rules of a decision tree -
```

```
print(tree.export_text(model,feature_names=feature_names,show_weights=True))
```

```
|--- Income <= 92.50
|   |--- CCAvg <= 2.95
|   |   |--- Securities_Account_1 <= 0.50
|   |   |   |--- weights: [218.40, 0.00] class: 0
|   |   |--- Securities_Account_1 >  0.50
|   |   |   |--- weights: [25.10, 0.00] class: 0
|   |--- CCAvg >  2.95
|   |   |--- CD_Account_1 <= 0.50
|   |   |   |--- CCAvg <= 3.95
|   |   |   |   |--- Mortgage <= 102.50
|   |   |   |   |   |--- CCAvg <= 3.05
|   |   |   |   |   |   |--- weights: [1.50, 0.00] class: 0
|   |   |   |   |   |--- CCAvg >  3.05
|   |   |   |   |   |   |--- Family_4 <= 0.50
|   |   |   |   |   |   |   |--- Income <= 67.00
|   |   |   |   |   |   |   |   |--- weights: [0.80, 0.00] class: 0
|   |   |   |   |   |   |   |--- Income >  67.00
|   |   |   |   |   |   |   |   |--- ZIPCodeZone_95 <= 0.50
|   |   |   |   |   |   |   |   |   |--- Income <= 84.00
|   |   |   |   |   |   |   |   |   |   |--- Securities_Account_1 <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 5
|   |   |   |   |   |   |   |   |   |   |--- Securities_Account_1 >  0.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- Income >  84.00
|   |   |   |   |   |   |   |   |   |   |--- ZIPCodeZone_94 <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.50, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- ZIPCodeZone_94 >  0.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |--- ZIPCodeZone_95 >  0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.40, 0.00] class: 0
|   |   |   |   |   |   |--- Family_4 >  0.50
|   |   |   |   |   |   |   |--- weights: [1.30, 0.00] class: 0
|   |   |   |   |--- Mortgage >  102.50
|   |   |   |   |   |--- Education_2 <= 0.50
|   |   |   |   |   |   |--- weights: [1.90, 0.00] class: 0
|   |   |   |   |   |--- Education_2 >  0.50
|   |   |   |   |   |   |--- weights: [0.20, 0.00] class: 0
|   |   |   |--- CCAvg >  3.95
|   |   |   |   |--- ZIPCodeZone_93 <= 0.50
|   |   |   |   |   |--- weights: [3.90, 0.00] class: 0
|   |   |   |   |--- ZIPCodeZone_93 >  0.50
|   |   |   |   |   |--- weights: [0.30, 0.00] class: 0
|   |   |--- CD_Account_1 >  0.50
|   |   |   |--- weights: [0.00, 4.50] class: 1
|--- Income >  92.50
|   |--- Education_3 <= 0.50
|   |   |--- Education_2 <= 0.50
|   |   |   |--- Family_3 <= 0.50
|   |   |   |   |--- Family_4 <= 0.50
|   |   |   |   |   |--- Income <= 103.50
|   |   |   |   |   |   |--- CCAvg <= 3.21
|   |   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |   |   |   |--- CCAvg >  3.21
|   |   |   |   |   |   |   |--- Experience <= 15.00
|   |   |   |   |   |   |   |   |--- weights: [0.30, 0.00] class: 0
|   |   |   |   |   |   |   |--- Experience >  15.00
|   |   |   |   |   |   |   |   |--- Income <= 97.00
|   |   |   |   |   |   |   |   |   |--- weights: [0.20, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- Income >  97.00
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.70] class: 1
|   |   |   |   |   |--- Income >  103.50
|   |   |   |   |   |   |--- ZIPCodeZone_96 <= 0.50
```

```
|   |   |   |   |   |   |   |   |--- weights: [42.90, 0.00] class: 0
|   |   |   |   |   |   |   |--- ZIPCodeZone_96 >  0.50
|   |   |   |   |   |   |   |   |--- weights: [0.40, 0.00] class: 0
|   |   |   |   |   |--- Family_4 >  0.50
|   |   |   |   |   |   |--- Income <= 93.50
|   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |--- Income >  93.50
|   |   |   |   |   |   |   |--- Income <= 102.00
|   |   |   |   |   |   |   |   |--- Income <= 97.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 0.90] class: 1
|   |   |   |   |   |   |   |   |--- Income >  97.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |   |--- Income >  102.00
|   |   |   |   |   |   |   |   |--- weights: [0.00, 17.10] class: 1
|   |   |   |   |--- Family_3 >  0.50
|   |   |   |   |   |--- Income <= 108.50
|   |   |   |   |   |   |--- weights: [1.10, 0.00] class: 0
|   |   |   |   |   |--- Income >  108.50
|   |   |   |   |   |   |--- Age <= 26.00
|   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |--- Age >  26.00
|   |   |   |   |   |   |   |--- Income <= 118.00
|   |   |   |   |   |   |   |   |--- Online_1 <= 0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.80] class: 1
|   |   |   |   |   |   |   |   |--- Online_1 >  0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.20, 0.00] class: 0
|   |   |   |   |   |   |   |--- Income >  118.00
|   |   |   |   |   |   |   |   |--- weights: [0.00, 29.70] class: 1
|   |   |--- Education_2 >  0.50
|   |   |   |--- Income <= 110.50
|   |   |   |   |--- CCAvg <= 2.90
|   |   |   |   |   |--- Income <= 106.50
|   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |   |   |--- Income >  106.50
|   |   |   |   |   |   |--- Experience <= 27.00
|   |   |   |   |   |   |   |--- ZIPCodeZone_95 <= 0.50
|   |   |   |   |   |   |   |   |--- weights: [0.40, 0.00] class: 0
|   |   |   |   |   |   |   |--- ZIPCodeZone_95 >  0.50
|   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |--- Experience >  27.00
|   |   |   |   |   |   |   |--- Family_4 <= 0.50
|   |   |   |   |   |   |   |   |--- ZIPCodeZone_94 <= 0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- ZIPCodeZone_94 >  0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |   |--- Family_4 >  0.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 0.90] class: 1
|   |   |   |   |--- CCAvg >  2.90
|   |   |   |   |   |--- Experience <= 30.00
|   |   |   |   |   |   |--- Securities_Account_1 <= 0.50
|   |   |   |   |   |   |   |--- weights: [0.00, 4.50] class: 1
|   |   |   |   |   |   |--- Securities_Account_1 >  0.50
|   |   |   |   |   |   |   |--- weights: [0.00, 0.90] class: 1
|   |   |   |   |   |--- Experience >  30.00
|   |   |   |   |   |   |--- weights: [0.20, 0.00] class: 0
|   |   |   |--- Income >  110.50
|   |   |   |   |--- Income <= 116.50
|   |   |   |   |   |--- Mortgage <= 126.25
|   |   |   |   |   |   |--- Age <= 60.50
|   |   |   |   |   |   |   |--- CCAvg <= 1.20
|   |   |   |   |   |   |   |   |--- weights: [0.20, 0.00] class: 0
|   |   |   |   |   |   |   |--- CCAvg >  1.20
|   |   |   |   |   |   |   |   |--- ZIPCodeZone_95 <= 0.50
|   |   |   |   |   |   |   |   |   |--- CCAvg <= 2.65
|   |   |   |   |   |   |   |   |   |   |--- Experience <= 15.00
```

```
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.80] class: 1
|   |   |   |   |   |   |   |   |   |   |--- Experience >  15.00
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.20, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- CCAvg >  2.65
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 4.50] class: 1
|   |   |   |   |   |   |   |   |--- ZIPCodeZone_95 >  0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |--- Age >  60.50
|   |   |   |   |   |   |   |--- weights: [0.30, 0.00] class: 0
|   |   |   |   |   |--- Mortgage >  126.25
|   |   |   |   |   |   |--- weights: [0.40, 0.00] class: 0
|   |   |   |   |--- Income >  116.50
|   |   |   |   |   |--- weights: [0.00, 97.20] class: 1
|   |--- Education_3 >  0.50
|   |   |--- Income <= 116.50
|   |   |   |--- CCAvg <= 2.35
|   |   |   |   |--- Mortgage <= 236.00
|   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |   |--- Mortgage >  236.00
|   |   |   |   |   |--- CCAvg <= 1.25
|   |   |   |   |   |   |--- Experience <= 14.00
|   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |--- Experience >  14.00
|   |   |   |   |   |   |   |--- weights: [0.00, 1.80] class: 1
|   |   |   |   |   |--- CCAvg >  1.25
|   |   |   |   |   |   |--- CCAvg <= 1.80
|   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |--- CCAvg >  1.80
|   |   |   |   |   |   |   |--- weights: [0.30, 0.00] class: 0
|   |   |   |--- CCAvg >  2.35
|   |   |   |   |--- Age <= 64.00
|   |   |   |   |   |--- CCAvg <= 2.95
|   |   |   |   |   |   |--- ZIPCodeZone_92 <= 0.50
|   |   |   |   |   |   |   |--- CreditCard_1 <= 0.50
|   |   |   |   |   |   |   |   |--- weights: [0.50, 0.00] class: 0
|   |   |   |   |   |   |   |--- CreditCard_1 >  0.50
|   |   |   |   |   |   |   |   |--- weights: [0.20, 0.00] class: 0
|   |   |   |   |   |   |--- ZIPCodeZone_92 >  0.50
|   |   |   |   |   |   |   |--- weights: [0.00, 1.80] class: 1
|   |   |   |   |   |--- CCAvg >  2.95
|   |   |   |   |   |   |--- ZIPCodeZone_92 <= 0.50
|   |   |   |   |   |   |   |--- Mortgage <= 172.00
|   |   |   |   |   |   |   |   |--- CD_Account_1 <= 0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 12.60] class: 1
|   |   |   |   |   |   |   |   |--- CD_Account_1 >  0.50
|   |   |   |   |   |   |   |   |   |--- Income <= 98.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 0.90] class: 1
|   |   |   |   |   |   |   |   |   |--- Income >  98.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |   |--- Mortgage >  172.00
|   |   |   |   |   |   |   |   |--- Experience <= 13.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.30, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- Experience >  13.50
|   |   |   |   |   |   |   |   |   |--- Mortgage <= 199.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- Mortgage >  199.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.70] class: 1
|   |   |   |   |   |   |--- ZIPCodeZone_92 >  0.50
|   |   |   |   |   |   |   |--- Mortgage <= 126.25
|   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |   |--- Mortgage >  126.25
|   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |--- Age >  64.00
|   |   |   |   |   |--- ZIPCodeZone_95 <= 0.50
|   |   |   |   |   |   |--- weights: [0.20, 0.00] class: 0
```

```
|   |   |   |   |   |--- ZIPCodeZone_95 >  0.50
|   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |--- Income >  116.50
|   |   |   |   |--- weights: [0.00, 102.60] class: 1
```

Identify the key variables

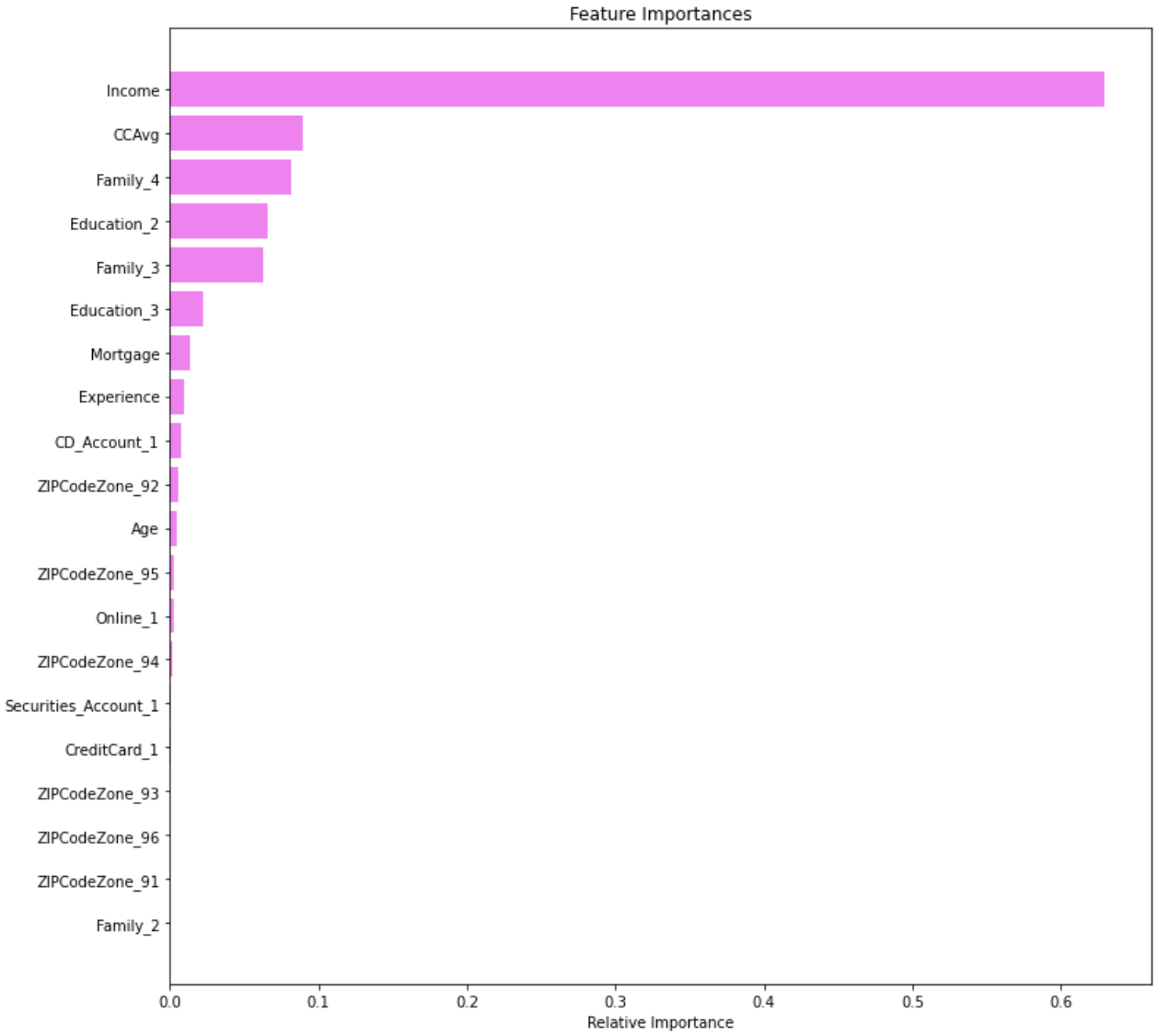
In [526…

```
# importance of features in the tree building ( The importance of a feature is computed
#(normalized) total reduction of the criterion brought by that feature. It is also know

print (pd.DataFrame(model.feature_importances_, columns = ["Imp"], index = X_train.colu
```

```
                       Imp
Income                0.63
CCAvg                 0.09
Family_4              0.08
Education_2           0.07
Family_3              0.06
Education_3           0.02
Mortgage              0.01
Experience            0.01
CD_Account_1          0.01
ZIPCodeZone_92        0.01
Age                   0.00
ZIPCodeZone_95        0.00
Online_1              0.00
ZIPCodeZone_94        0.00
Securities_Account_1  0.00
CreditCard_1          0.00
ZIPCodeZone_93        0.00
ZIPCodeZone_96        0.00
Family_2              0.00
ZIPCodeZone_91        0.00
```

In [526…

```
importances = model.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



- According to the decision tree model, `Income` is the most important variable for predicting the `Personal_Loan` .

Reducing over fitting - with Pre-Pruning

Using GridSearch for Hyperparameter tuning of our tree model

In [526…

```
from sklearn.model_selection import GridSearchCV

# Choose the type of classifier.
estimator = DecisionTreeClassifier(random_state=1,class_weight = {0:.10,1:.90})

# Grid of parameters to choose from
parameters = {
            'max_depth': np.arange(1,10),
            'criterion': ['entropy','gini'],
            'splitter': ['best','random'],
            'min_impurity_decrease': [0.000001,0.00001,0.0001],
            'max_features': ['log2','sqrt']
             }

# Type of scoring used to compare parameter combinations
```

```
scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(estimator, parameters, scoring=scorer,cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
estimator = grid_obj.best_estimator_

# Fit the best algorithm to the data.
estimator.fit(X_train, y_train)
```

Out[526…
```
DecisionTreeClassifier(class_weight={0: 0.1, 1: 0.9}, max_depth=5,
                       max_features='log2', min_impurity_decrease=1e-06,
                       random_state=1)
```

In [526…
```
make_confusion_matrix(estimator,y_test)
```

In [527…
```
get_recall_score(estimator)
```

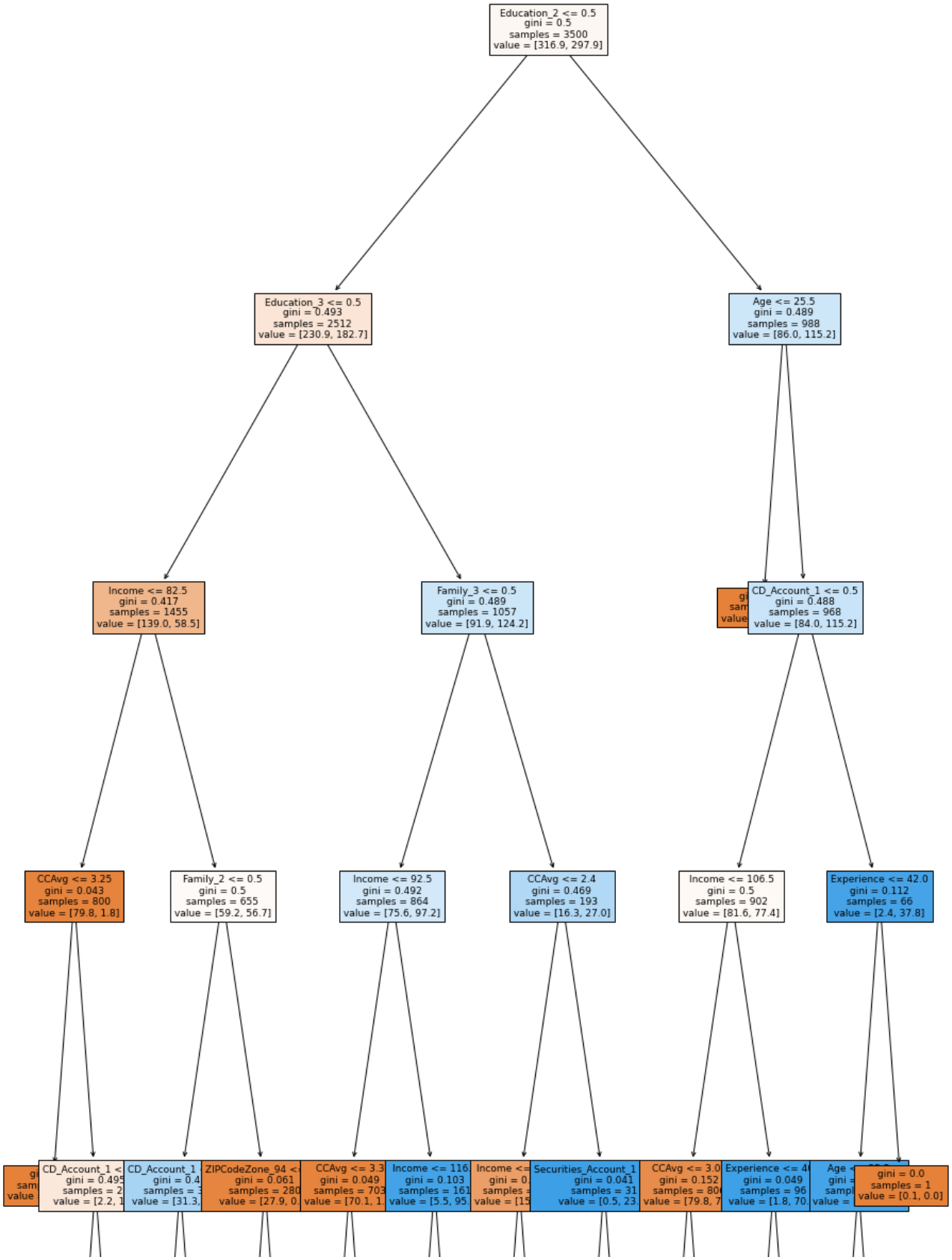
```
Recall on training set :  0.9879154078549849
Recall on test set :  0.9261744966442953
```

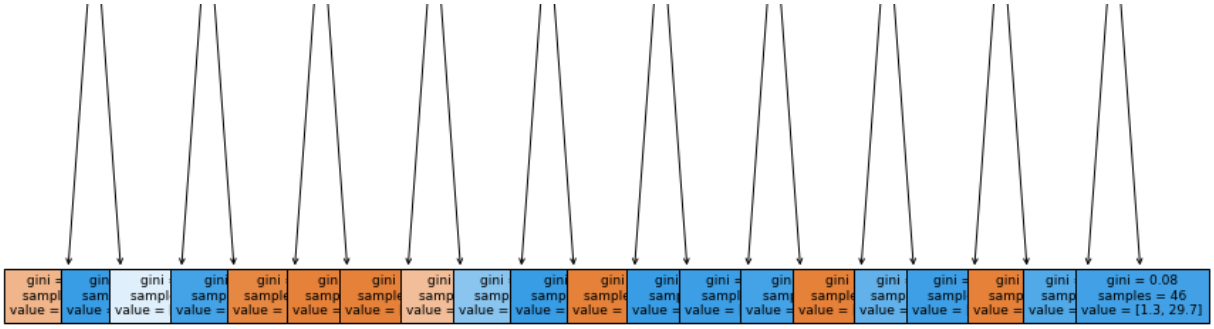
- Recall has improved for both train and test set after hyperparameter tuning and we have a generalized model.

Visualizing the Decision Tree - after reducing overfitting (pre-pruning)

In [527…
```
plt.figure(figsize=(15,30))
out = tree.plot_tree(estimator,feature_names=feature_names,filled=True,fontsize=9,node_
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
```

```
        arrow.set_edgecolor('black')
        arrow.set_linewidth(1)
plt.show()
```





In [527…

```
# Text report showing the rules of a decision tree -

print(tree.export_text(estimator,feature_names=feature_names,show_weights=True))
```

```
|--- Education_2 <= 0.50
|   |--- Education_3 <= 0.50
|   |   |--- Income <= 82.50
|   |   |   |--- CCAvg <= 3.25
|   |   |   |   |--- weights: [77.60, 0.00] class: 0
|   |   |   |--- CCAvg >  3.25
|   |   |   |   |--- CD_Account_1 <= 0.50
|   |   |   |   |   |--- weights: [2.20, 0.90] class: 0
|   |   |   |   |--- CD_Account_1 >  0.50
|   |   |   |   |   |--- weights: [0.00, 0.90] class: 1
|   |   |--- Income >  82.50
|   |   |   |--- Family_2 <= 0.50
|   |   |   |   |--- CD_Account_1 <= 0.50
|   |   |   |   |   |--- weights: [30.20, 36.00] class: 1
|   |   |   |   |--- CD_Account_1 >  0.50
|   |   |   |   |   |--- weights: [1.10, 19.80] class: 1
|   |   |   |--- Family_2 >  0.50
|   |   |   |   |--- ZIPCodeZone_94 <= 0.50
|   |   |   |   |   |--- weights: [20.60, 0.90] class: 0
|   |   |   |   |--- ZIPCodeZone_94 >  0.50
|   |   |   |   |   |--- weights: [7.30, 0.00] class: 0
|   |--- Education_3 >  0.50
|   |   |--- Family_3 <= 0.50
|   |   |   |--- Income <= 92.50
|   |   |   |   |--- CCAvg <= 3.30
|   |   |   |   |   |--- weights: [66.40, 0.00] class: 0
|   |   |   |   |--- CCAvg >  3.30
|   |   |   |   |   |--- weights: [3.70, 1.80] class: 0
|   |   |   |--- Income >  92.50
|   |   |   |   |--- Income <= 116.50
|   |   |   |   |   |--- weights: [5.50, 13.50] class: 1
|   |   |   |   |--- Income >  116.50
|   |   |   |   |   |--- weights: [0.00, 81.90] class: 1
|   |   |--- Family_3 >  0.50
|   |   |   |--- CCAvg <= 2.40
|   |   |   |   |--- Income <= 119.50
|   |   |   |   |   |--- weights: [15.80, 0.00] class: 0
|   |   |   |   |--- Income >  119.50
|   |   |   |   |   |--- weights: [0.00, 3.60] class: 1
|   |   |   |--- CCAvg >  2.40
|   |   |   |   |--- Securities_Account_1 <= 0.50
|   |   |   |   |   |--- weights: [0.30, 19.80] class: 1
|   |   |   |   |--- Securities_Account_1 >  0.50
```

```
|   |   |   |   |   |--- weights: [0.20, 3.60] class: 1
|--- Education_2 >  0.50
|   |--- Age <= 25.50
|   |   |--- weights: [2.00, 0.00] class: 0
|   |--- Age >  25.50
|   |   |--- CD_Account_1 <= 0.50
|   |   |   |--- Income <= 106.50
|   |   |   |   |--- CCAvg <= 3.05
|   |   |   |   |   |--- weights: [78.30, 0.00] class: 0
|   |   |   |   |--- CCAvg >  3.05
|   |   |   |   |   |--- weights: [1.50, 7.20] class: 1
|   |   |   |--- Income >  106.50
|   |   |   |   |--- Experience <= 40.50
|   |   |   |   |   |--- weights: [1.70, 70.20] class: 1
|   |   |   |   |--- Experience >  40.50
|   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |--- CD_Account_1 >  0.50
|   |   |   |--- Experience <= 42.00
|   |   |   |   |--- Age <= 35.50
|   |   |   |   |   |--- weights: [1.00, 8.10] class: 1
|   |   |   |   |--- Age >  35.50
|   |   |   |   |   |--- weights: [1.30, 29.70] class: 1
|   |   |   |--- Experience >  42.00
|   |   |   |   |--- weights: [0.10, 0.00] class: 0
```

Identify the key variables

In [527…

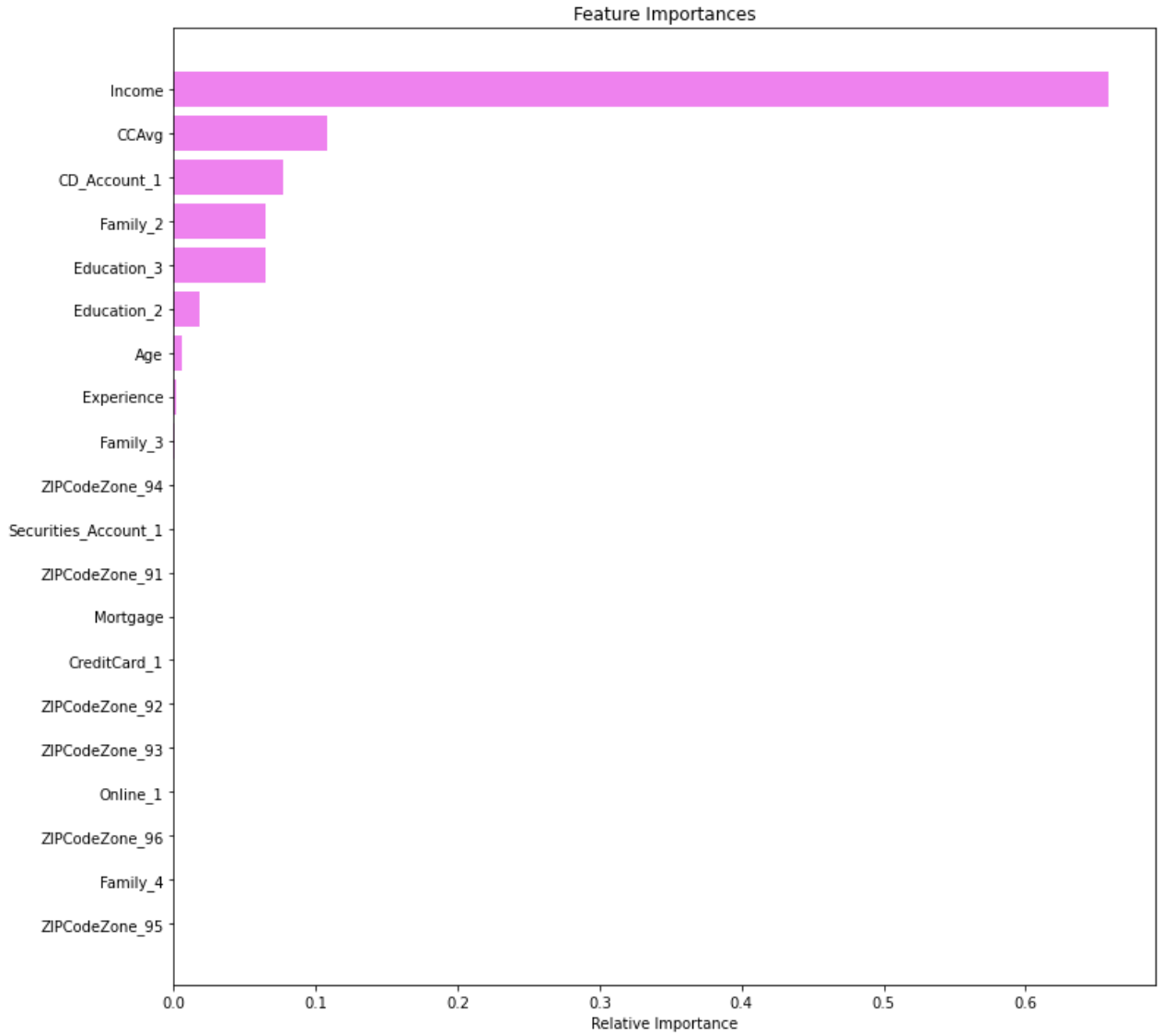
```
# importance of features in the tree building ( The importance of a feature is computed
#(normalized) total reduction of the 'criterion' brought by that feature. It is also kn

print (pd.DataFrame(estimator.feature_importances_, columns = ["Imp"], index = X_train.

#Here we will see that importance of features has increased
```

```
                       Imp
Income                0.66
CCAvg                 0.11
CD_Account_1          0.08
Family_2              0.06
Education_3           0.06
Education_2           0.02
Age                   0.01
Experience            0.00
Family_3              0.00
ZIPCodeZone_94        0.00
Securities_Account_1  0.00
Online_1              0.00
Family_4              0.00
ZIPCodeZone_96        0.00
ZIPCodeZone_95        0.00
ZIPCodeZone_93        0.00
ZIPCodeZone_92        0.00
ZIPCodeZone_91        0.00
Mortgage              0.00
CreditCard_1          0.00
```

In [527…

```
importances = estimator.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
```

```
plt.xlabel('Relative Importance')
plt.show()
```



- After reducing the overfitting of the model, `CCAvg` is the most important variable for predicting the `Personal_Loan`, followed by `Income`.

Total impurity of leaves vs effective alphas of pruned tree

To get an idea of what values of `ccp_alpha` could be appropriate, `DecisionTreeClassifier.cost_complexity_pruning_path` returns the effective alphas and the corresponding total leaf impurities at each step of the pruning process. As alpha increases, more of the tree is pruned, which increases the total impurity of its leaves.

In [527…

```
clf = DecisionTreeClassifier(random_state=1,class_weight = {0:0.10,1:0.90})
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

pd.DataFrame(path)
```

Out[527…

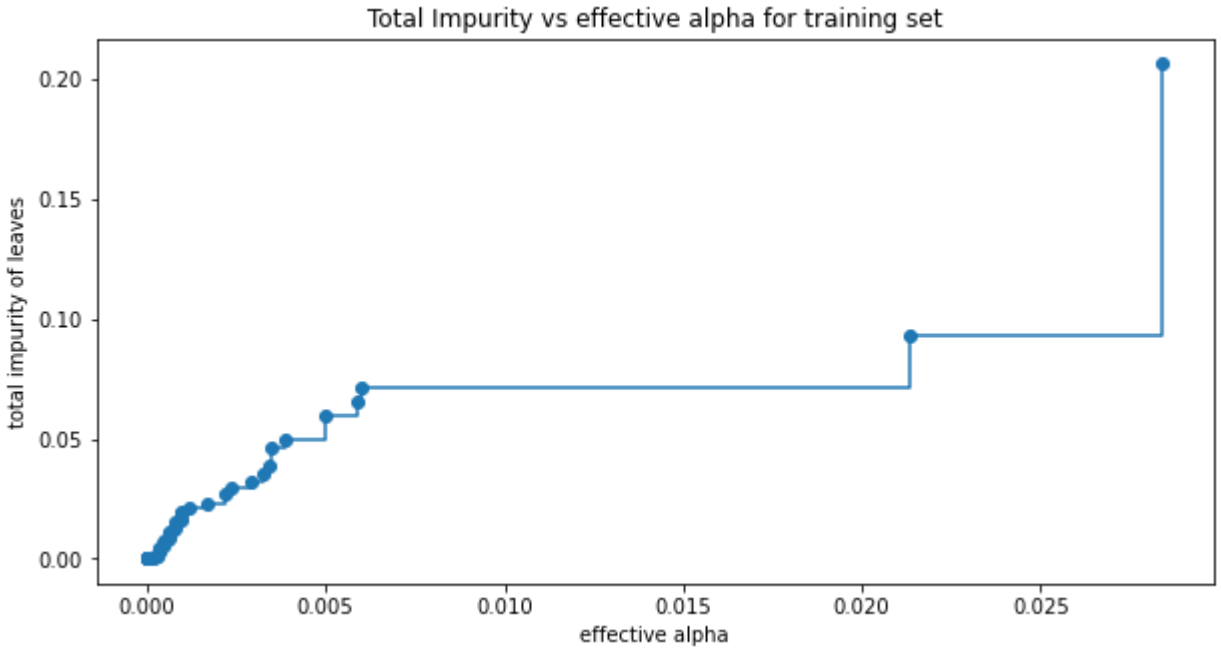
	ccp_alphas	impurities

	ccp_alphas	impurities
0	0.00	-0.00
1	0.00	-0.00
2	0.00	-0.00
3	0.00	-0.00
4	0.00	-0.00
5	0.00	-0.00
6	0.00	-0.00
7	0.00	-0.00
8	0.00	-0.00
9	0.00	-0.00
10	0.00	-0.00
11	0.00	-0.00
12	0.00	0.00
13	0.00	0.00
14	0.00	0.00
15	0.00	0.00
16	0.00	0.00
17	0.00	0.00
18	0.00	0.00
19	0.00	0.00
20	0.00	0.00
21	0.00	0.00
22	0.00	0.01
23	0.00	0.01
24	0.00	0.01
25	0.00	0.01
26	0.00	0.01
27	0.00	0.01
28	0.00	0.01
29	0.00	0.01
30	0.00	0.01
31	0.00	0.01
32	0.00	0.02

	ccp_alphas	impurities
33	0.00	0.02
34	0.00	0.02
35	0.00	0.02
36	0.00	0.02
37	0.00	0.02
38	0.00	0.03
39	0.00	0.03
40	0.00	0.03
41	0.00	0.04
42	0.00	0.04
43	0.00	0.05
44	0.00	0.05
45	0.00	0.06
46	0.01	0.07
47	0.01	0.07
48	0.02	0.09
49	0.03	0.21
50	0.29	0.50

In [527…

```
fig, ax = plt.subplots(figsize=(10,5))
ax.plot(ccp_alphas[:-1], impurities[:-1], marker='o', drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for training set")
plt.show()
```

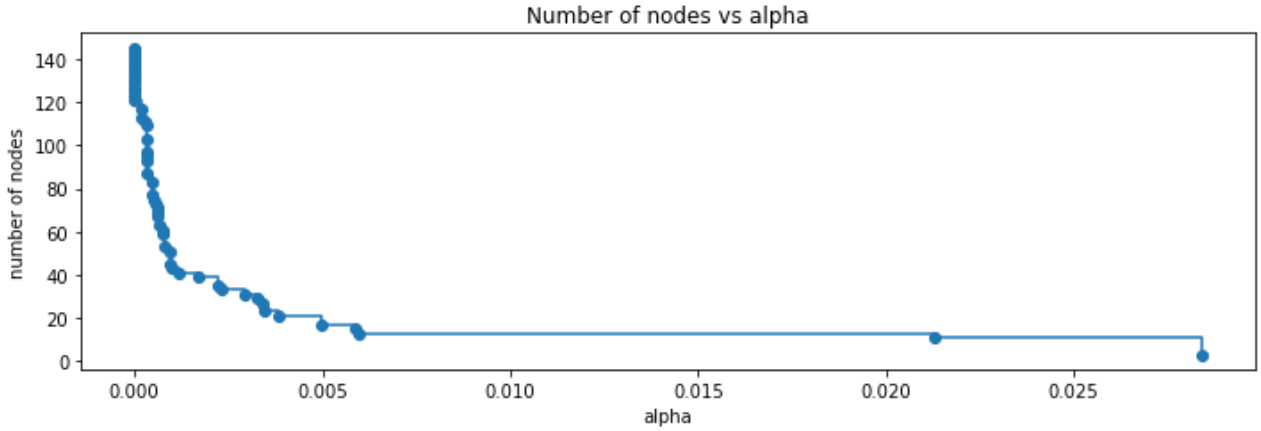


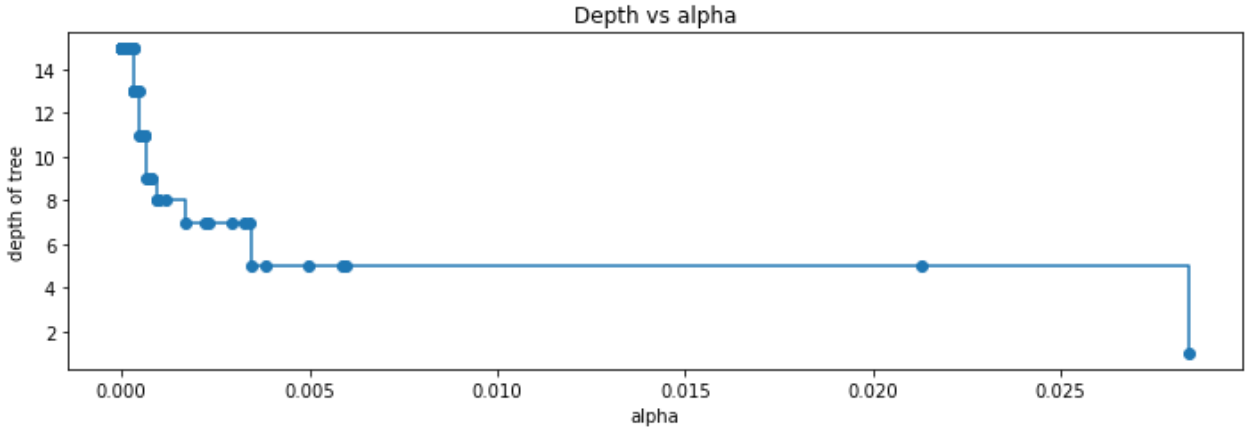
We will train a decision tree using the effective alphas.

In [527…
```
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=1, ccp_alpha=ccp_alpha,class_weight = {0:
    clf.fit(X_train, y_train)
    clfs.append(clf)
print("Number of nodes in the last tree is: {} with ccp_alpha: {}".format(
      clfs[-1].tree_.node_count, ccp_alphas[-1]))
```

```
Number of nodes in the last tree is: 1 with ccp_alpha: 0.2928785401980034
```

In [527…
```
clfs = clfs[:-1]
ccp_alphas = ccp_alphas[:-1]

node_counts = [clf.tree_.node_count for clf in clfs]
depth = [clf.tree_.max_depth for clf in clfs]
fig, ax = plt.subplots(2, 1,figsize=(10,7))
ax[0].plot(ccp_alphas, node_counts, marker='o', drawstyle="steps-post")
ax[0].set_xlabel("alpha")
ax[0].set_ylabel("number of nodes")
ax[0].set_title("Number of nodes vs alpha")
ax[1].plot(ccp_alphas, depth, marker='o', drawstyle="steps-post")
ax[1].set_xlabel("alpha")
ax[1].set_ylabel("depth of tree")
ax[1].set_title("Depth vs alpha")
fig.tight_layout()
```



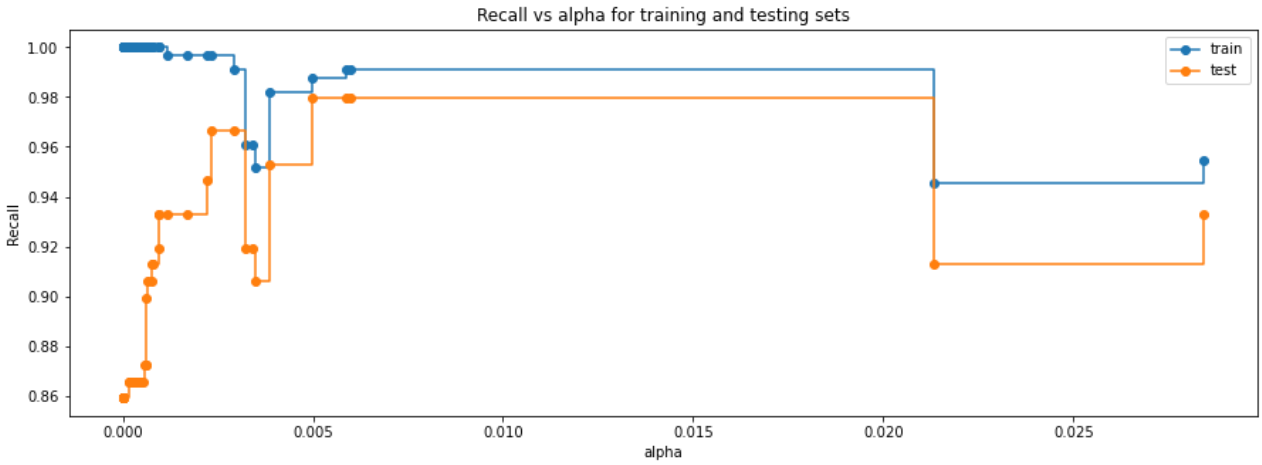


In [527…
```
recall_train=[]
for clf in clfs:
    pred_train3=clf.predict(X_train)
    values_train=metrics.recall_score(y_train,pred_train3)
    recall_train.append(values_train)
```

In [528…
```
recall_test=[]
for clf in clfs:
    pred_test3=clf.predict(X_test)
    values_test=metrics.recall_score(y_test,pred_test3)
    recall_test.append(values_test)
```

In [528…
```
train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]
```

In [528…
```
fig, ax = plt.subplots(figsize=(15,5))
ax.set_xlabel("alpha")
ax.set_ylabel("Recall")
ax.set_title("Recall vs alpha for training and testing sets")
ax.plot(ccp_alphas, recall_train, marker='o', label="train",
        drawstyle="steps-post",)
ax.plot(ccp_alphas, recall_test, marker='o', label="test",
        drawstyle="steps-post")
ax.legend()
plt.show()
```



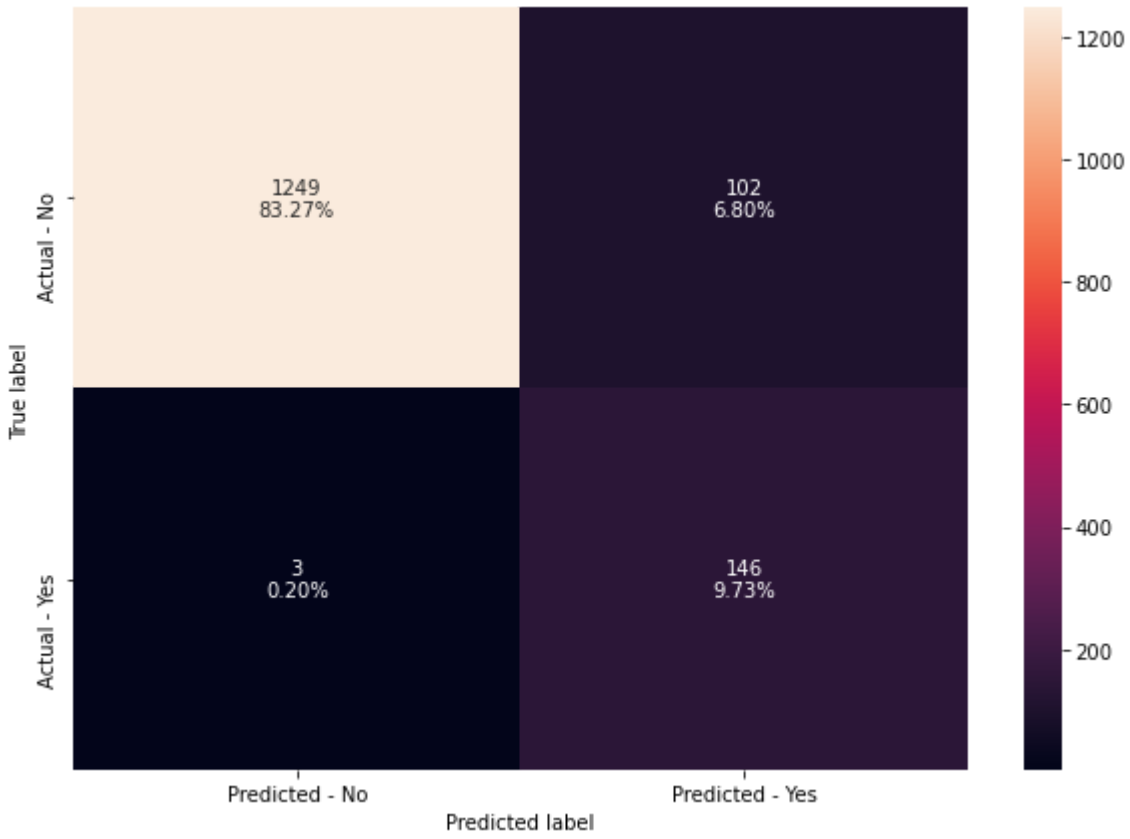
- Maximum value of Recall is at 0.031 alpha, but if we choose decision tree will only have a root node and we would lose the business rules, instead we can choose alpha 0.006 retaining information and getting higher recall

Creating model with 0.006 ccp_alpha

In [528…
```
best_model2 = DecisionTreeClassifier(ccp_alpha=0.006,
                        class_weight={0: 0.10, 1: 0.90}, random_state=1)
best_model2.fit(X_train, y_train)
```

Out[528…
```
DecisionTreeClassifier(ccp_alpha=0.006, class_weight={0: 0.1, 1: 0.9},
                       random_state=1)
```

In [528…
```
make_confusion_matrix(best_model2,y_test)
```



- We are able to identify more True positives.Customers that are actually going to converto to Asset Customer; than any other model.

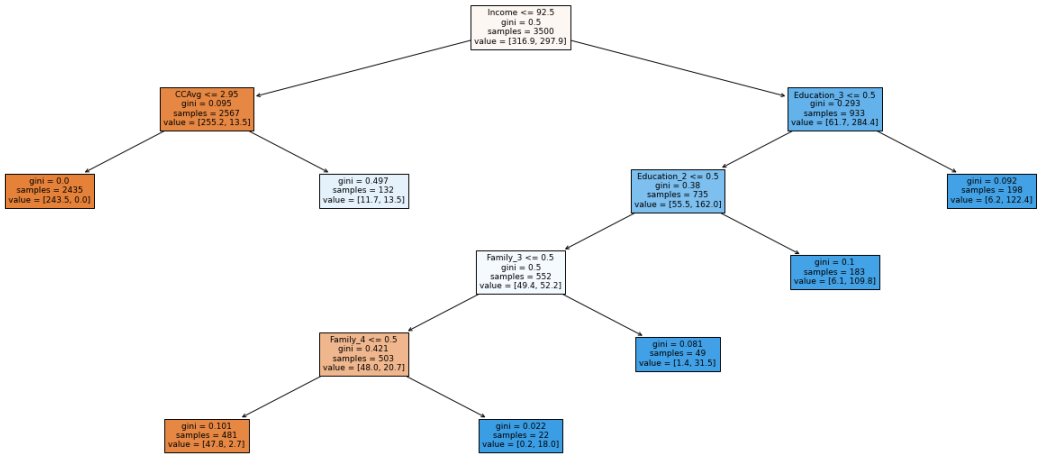
In [528…
```
get_recall_score(best_model2)
```

```
Recall on training set :  0.9909365558912386
Recall on test set :  0.9798657718120806
```

- The results have improved from the initial model and we have got higher recall (0.81 vs 0.97) than the hyperparameter tuned model and generalized decision tree. Having comparable performance on test set.

Visualizing the Decision Tree - with post-pruning (0.006 ccp_alpha)

In [528…
```
plt.figure(figsize=(25,10))

out = tree.plot_tree(best_model2,feature_names=feature_names,filled=True,fontsize=9,nod
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor('black')
        arrow.set_linewidth(1)
plt.show()
```



In [528…
```
# Text report showing the rules of a decision tree -

print(tree.export_text(best_model2,feature_names=feature_names,show_weights=True))
```

```
|--- Income <= 92.50
|   |--- CCAvg <= 2.95
|   |   |--- weights: [243.50, 0.00] class: 0
|   |--- CCAvg >  2.95
|   |   |--- weights: [11.70, 13.50] class: 1
|--- Income >  92.50
|   |--- Education_3 <= 0.50
|   |   |--- Education_2 <= 0.50
|   |   |   |--- Family_3 <= 0.50
|   |   |   |   |--- Family_4 <= 0.50
|   |   |   |   |   |--- weights: [47.80, 2.70] class: 0
|   |   |   |   |--- Family_4 >  0.50
|   |   |   |   |   |--- weights: [0.20, 18.00] class: 1
|   |   |   |--- Family_3 >  0.50
|   |   |   |   |--- weights: [1.40, 31.50] class: 1
```

```
|   |   |--- Education_2 >  0.50
|   |   |   |--- weights: [6.10, 109.80] class: 1
|   |--- Education_3 >  0.50
|   |   |--- weights: [6.20, 122.40] class: 1
```

Identify the key variables

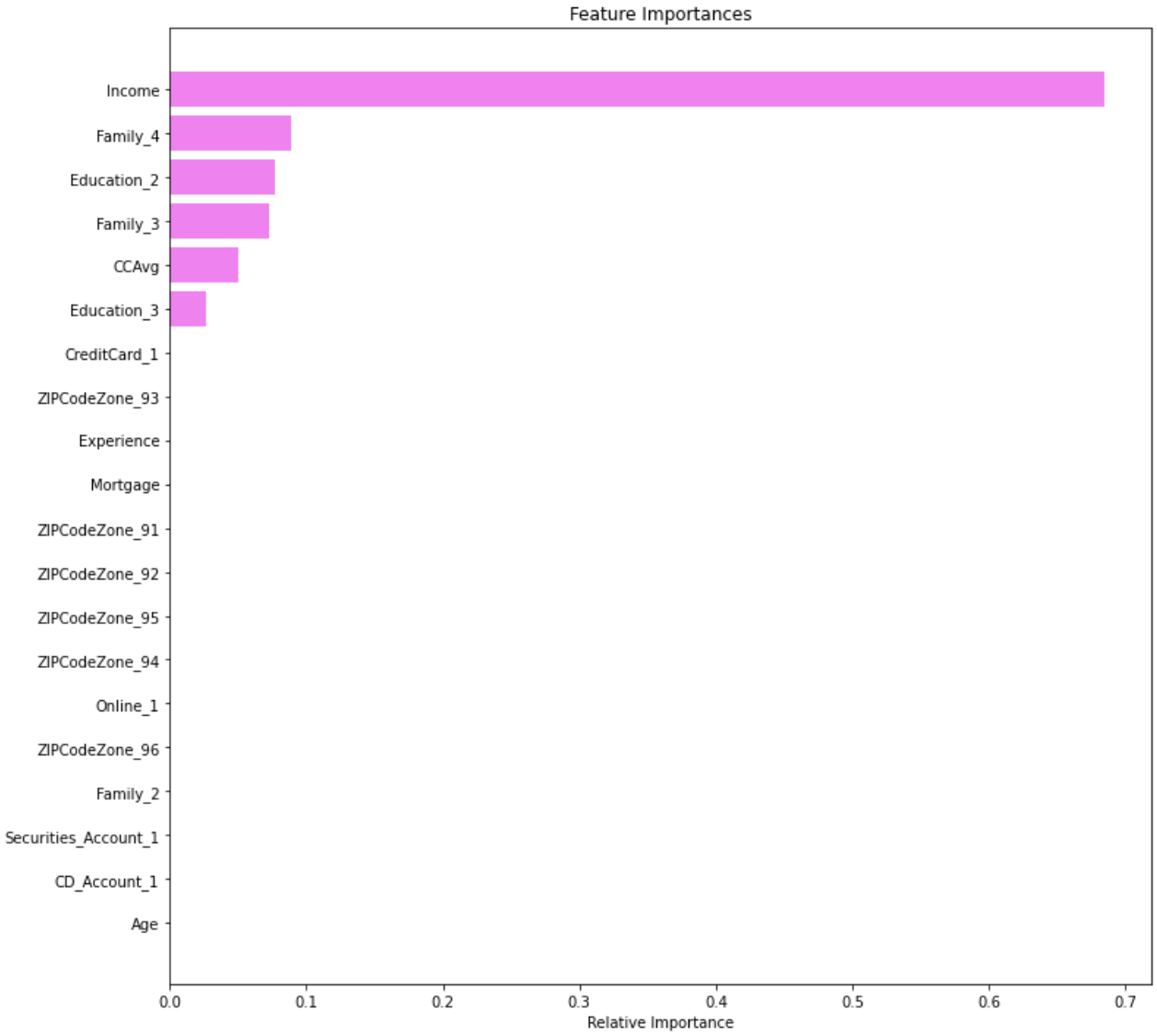
In [528…

```
# importance of features in the tree building ( The importance of a feature is computed
#(normalized) total reduction of the 'criterion' brought by that feature. It is also kn

print (pd.DataFrame(best_model2.feature_importances_, columns = ["Imp"], index = X_trai
```

```
                       Imp
Income                0.68
Family_4              0.09
Education_2           0.08
Family_3              0.07
CCAvg                 0.05
Education_3           0.03
Age                   0.00
Online_1              0.00
CD_Account_1          0.00
Securities_Account_1  0.00
Family_2              0.00
ZIPCodeZone_96        0.00
Experience            0.00
ZIPCodeZone_95        0.00
ZIPCodeZone_94        0.00
ZIPCodeZone_93        0.00
ZIPCodeZone_92        0.00
ZIPCodeZone_91        0.00
Mortgage              0.00
CreditCard_1          0.00
```

In [528…

```
importances = best_model2.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



- `Income` is the most important features to predict customer conversion to Asset Customer.

Comparing all the decision tree models

In [529…

```
comparison_frame = pd.DataFrame({'Model':['Initial decision tree model','Decision treee
                                          'Decision tree with post-pruning'], 'Train_Re
comparison_frame
```

Out[529…

	Model	Train_Recall	Test_Recall
0	Initial decision tree model	1.00	0.30
1	Decision treee with hyperparameter tuning	0.83	0.81
2	Decision tree with post-pruning	0.89	0.86

Recommendations

- Highest performing Logistic Regression model is : `* Logistic Regression with feature elimination`
- Logistic Regression most important features are :

    - Education
    - CD_Account
- Highest performing Desicion Tree model is : `Decision tree with post-pruning`
- Decision Tree most important features are :
    - Income
    - Family_4
- It is predicted that customers with higher education and that already have a CD_Account are more likely to convert to Asset Customers.
- It is predicted that Customers with high Income and on average are a family of 4 are more likley to conert to Asset Customers.

In [ ]:

End of File